1. Introducción a Lenguajes y Paradigmas de Programación
    1.1. Introducción

- Lenguajes de Programación:

Los lenguajes de programación son conjuntos de instrucciones y reglas utilizados para escribir programas informáticos. Estas instrucciones permiten a los desarrolladores comunicarse con las computadoras y especificar cómo realizar tareas y procesos. Los lenguajes de programación pueden ser de diferentes tipos, desde aquellos diseñados para propósitos generales hasta aquellos especializados en áreas específicas como el desarrollo web, científico o de sistemas.

- Paradigmas de Programación:

Los paradigmas de programación son enfoques o estilos distintos para resolver problemas mediante la codificación. Cada paradigma tiene sus propias reglas y convenciones, y se basa en diferentes conceptos teóricos y filosóficos.

01. **Imperativo**:

Este paradigma se centra en la descripción de los pasos o procedimientos necesarios para alcanzar un resultado. Los programas imperativos están compuestos por secuencias de comandos que modifican el estado de las variables y la memoria de la computadora.

02. **Declarativo**:

A diferencia del paradigma imperativo, el enfoque declarativo se centra en describir qué se quiere lograr en lugar de cómo lograrlo. Los programas declarativos especifican las relaciones entre diferentes entidades y permiten que el sistema determine la mejor manera de llegar al resultado deseado.

03. **Orientado a Objetos**:

Este paradigma se basa en la representación de entidades como objetos que tienen propiedades (atributos) y comportamientos (métodos). Los programas orientados a objetos están estructurados en torno a la interacción entre estos objetos, lo que facilita la reutilización de código y la modularidad.

04. **Funcional**:

En este paradigma, los programas se construyen mediante la composición de funciones puras, es decir, funciones que no tienen efectos secundarios y producen el mismo resultado cada vez que se invocan con los mismos argumentos. El enfoque funcional se basa en la aplicación de funciones matemáticas y la programación basada en expresiones.

- Lenguajes de alto y bajo nivel:

Diferencia	
Alto Nivel	**Bajo Nivel**
Estos lenguajes están más alejados del lenguaje de máquina y son más cercanos al lenguaje humano. Utilizan abstracciones y estructuras de datos complejas, lo que facilita la escritura y comprensión del	Estos lenguajes están más cerca del lenguaje de máquina y son menos comprensibles para los humanos. Están diseñados para interactuar directamente con el hardware de la computadora y

código. Suelen ser independientes de la arquitectura del hardware, lo que significa que un programa escrito en un lenguaje de alto nivel puede ejecutarse en diferentes tipos de computadoras sin necesidad de modificaciones significativas.

suelen ser específicos de una arquitectura de procesador. Los lenguajes de bajo nivel permiten un control más preciso sobre el hardware de la computadora y son más eficientes en términos de uso de recursos, pero escribir programas en ellos puede ser más complicado y propenso a errores.

Ventajas	
Alto Nivel	**Bajo Nivel**
- **Mayor portabilidad**: Los programas escritos en lenguaje de alto nivel pueden ejecutarse en diferentes plataformas sin necesidad de modificaciones importantes. - **Mayor legibilidad y mantenibilidad**: Debido a que se asemejan más al lenguaje humano, son fáciles de entender y modificar. - **Desarrollo rápido**: Permiten desarrollar aplicaciones en menos tiempo debido a su sintaxis simplificada y el uso de abstracciones.	- **Mayor control sobre el hardware**: Permiten acceder y manipular directamente los recursos del sistema, lo que los hace ideales para desarrollar software de bajo nivel como sistemas operativos o controladores de dispositivos. - **Mayor eficiencia en términos de rendimiento**: Debido a su cercanía con el hardware, los programas escritos en lenguaje de bajo nivel pueden ser más eficientes en términos de uso de recursos.

Desventajas	
Alto Nivel	**Bajo Nivel**
- **Menor control sobre el hardware**: No ofrecen el mismo nivel de control sobre el hardware de la computadora que los lenguajes de bajo nivel. - **Menor eficiencia en términos de rendimiento**: Los programas escritos en lenguajes de alto nivel pueden ser menos eficientes en términos de uso de recursos (memoria, CPU) en comparación con los escritos en lenguajes de bajo nivel.	- **Menor portabilidad**: Los programas escritos en lenguajes de bajo nivel suelen ser específicos de una arquitectura de procesador y pueden no ser fácilmente portables a otras plataformas. - **Mayor complejidad y propensión a errores**: Escribir programas en lenguajes de bajo nivel puede ser más difícil y propenso a errores debido a su sintaxis más compleja y la necesidad de gestionar directamente los recursos del sistema.

Ejemplos	
Alto Nivel	**Bajo Nivel**
- **Python**: Utilizado en desarrollo web, ciencia de	- **Ensamblador**: Utilizado en el desarrollo de sistemas

datos, automatización de tareas, entre otros.

- **Java**:
Ampliamente utilizado en desarrollo de aplicaciones empresariales, aplicaciones móviles (Android), sistemas embebidos, entre otros.
- **C#**:
Principalmente utilizado en desarrollo de aplicaciones Windows, juegos, y desarrollo de software empresarial.

operativos, controladores de dispositivos, y en aplicaciones que requieren un control preciso sobre el hardware.

- **C y C++**:
Ampliamente utilizado en el desarrollo de sistemas operativos, sistemas embebidos, aplicaciones de tiempo real, y en aplicaciones que requieren alta eficiencia y control directo sobre el hardware.

- Frameworks:

Los Frameworks son conjuntos de herramientas, bibliotecas y estándares que proporcionan una estructura para el desarrollo de aplicaciones. Los Frameworks permiten a los desarrolladores crear aplicaciones más rápidamente al proporcionar soluciones predefinidas para tareas comunes, como la gestión de bases de datos, la autenticación de usuarios y la generación de interfaces de usuario.

01. **Angular(JavaScript/TypeScript)**:

Framework de Google para construir aplicaciones web SPA con JavaScript o TypeScript.

02. **Next + React (JavaScript)**:

Biblioteca de Facebook para crear interfaces de usuario interactivas y dinámicas con JavaScript.

03. **Vue.js (JavaScript)**:

Framework progresivo para crear interfaces de usuario interactivas con JavaScript, conocido por su simplicidad y flexibilidad.

04. **Django (Python)**:

Framework de alto nivel para el desarrollo web con Python, conocido por su enfoque “baterías incluidas” y su rapidez de desarrollo.

05. **Ruby on Rails (Ruby)**:

Framework web de código abierto escrito en Ruby, famoso por su enfoque de “convención sobre configuración” y su rapidez de desarrollo.

06. **Spring Framework (Java)**:

Framework de aplicación empresarial para el desarrollo de aplicaciones Java, conocido por su modularidad y amplia gama de funcionalidades.

1.2. Intérpretes, Compiladores y Código Fuente

- Intérpretes:

Un intérprete es un programa que lee y ejecuta instrucciones escritas en un lenguaje de programación directamente, línea por línea. Convierte el código fuente en instrucciones ejecutables de forma inmediata. Ejemplos de lenguajes interpretados incluyen Python, Ruby y JavaScript.

- Compiladores:

Un compilador es un programa que traduce el código fuente escrito en un lenguaje de programación a un código de máquina (binario) entendible por la computadora. Este proceso se realiza en una etapa previa a la ejecución del programa. Ejemplos de lenguajes compilados incluye C, C + +, Java (a bytecode) y Rust.

- Intérprete vs. Compiladores:

01. **Tiempo de Ejecución**:

Los programas escritos en lenguajes interpretados son ejecutados línea por línea por el intérprete, mientras que los programas compilados son traducidos completamente antes de la ejecución, lo que puede resultar en una mayor eficiencia durante la ejecución.

02. **Portabilidad**:

Los programas interpretados suelen ser más portátiles ya que solo necesitan un intérprete para ejecutarse en diferentes plataformas, mientras que los programas compilados pueden necesitar ser compilados para cada plataforma específica.

03. **Detección de Errores**:

Los compiladores pueden detectar errores en el código durante la fase de compilación, mientras que los intérpretes detectan errores a medida que se ejecuta el programa.

- Código Fuente:

El código fuente es el conjunto de instrucciones escritas por un programador en un lenguaje de programación específico antes de ser traducido a un formato ejecutable.
Es la representación legible y comprensible por humanos del programa que se desea desarrollar.
El código fuente es fundamental en el desarrollo de software ya que es la base sobre la cual se construye el programa final.
Permite la colaboración entre programadores, la revisión de código, la depuración de errores y la mantenibilidad del software a lo largo del tiempo.

1.3. IDEs y editores de texto

- Editores de texto:

Los editores de texto son herramientas básicas para escribir y editar código fuente. A diferencia de los procesadores de texto, los editores de texto están diseñados específicamente para trabajar con código, lo que los hace más eficientes y adecuados para programadores.

01. Sublime Text:

Conocido por su velocidad y simplicidad, Sublime Text es altamente personalizable cuenta con una amplia gama de complementos disponibles.

02. Atom:

Desarrollado por GitHub, Atom es un editor de texto de código abierto que ofrece integración con Git y una comunidad activa de desarrolladores que crean complementos y temas.

03. Visual Studio Code (VS Code, VSC):
Desarrollado por Microsoft, VS Code es uno de los editores de texto más populares debido a su robusta funcionalidad, amplia gama de extensiones y soporte para múltiples lenguajes de programación.

- IDE

Los Entornos de Desarrollo Integrado (IDEs) son aplicaciones que proporcionan un conjunto completo de herramientas para el desarrollo de software en un solo paquete integrado. Estas herramientas incluyen editores de texto, compiladores, depuradores, administradores de proyectos y más.

01. **Elipse**:
Es una plataforma de desarrollo de código abierto ampliamente utilizada en varios dominios, incluido el desarrollo de aplicaciones Java.

02. **NetBeans**:
Desarrollado inicialmente por Sun Microsystems y luego adquirido por Oracle, NetBeans es conocido por su aporte para múltiples lenguajes de programación y su extensibilidad.

03. **IntelliJ IDEA**:
Es conocido por su excelente soporte para el desarrollo de aplicaciones Java, así como su integración con tecnologías como Spring Framework y Android. Está muy ligado a Android Studio.

1.4. Escritorios Remotos

- ¿Qué es un escritorio remoto y cómo funciona?

Un escritorio remoto es una tecnología que permite a un usuario acceder y controlar una computadora o dispositivo desde otro lugar, a través de una red, como Internet. En lugar de interactuar con el sistema operativo localmente, el usuario puede ver y manipular la interfaz gráfica de usuario (GUI) de la computadora remota como si estuviera frente a ella físicamente. Esto significa que puedes acceder a sus archivos, aplicaciones y recursos de software desde cualquier lugar con conexión a internet.

El funcionamiento básico de un escritorio remoto implica que el dispositivo local (cliente) se conecta a través de una red al dispositivo remoto (servidor) que está ejecutando un software de servidor de escritorio remoto. Este software captura la pantalla del servidor y envía las actualizaciones de la pantalla al cliente. A su vez, el cliente envía las entradas del usuario (como clics del mouse y pulsaciones de teclas) de regreso al servidor, que las ejecuta como si fueran entradas locales.

- Ventajas de utilizar escritorios remotos en entornos de trabajo colaborativos:

01. **Acceso remoto**:
Permite a los usuarios acceder a sus recursos informáticos desde cualquier lugar, en cualquier momento, lo que facilita el trabajo remoto y la colaboración entre equipos distribuidos geográficamente.

02. **Flexibilidad**:
Los usuarios pueden utilizar sus propios dispositivos para acceder al escritorio remoto, lo que les permite trabajar con sus herramientas preferidas desde cualquier lugar

03. **Centralización y gestión simplificada**:
Al tener todas las aplicaciones y datos almacenados en un servidor central, la gestión de software, actualizaciones y copias de seguridad se vuelve más sencilla y segura.

04. **Seguridad**:
Los datos permanecen en el servidor remoto, reduciendo el riesgo de pérdida o robo de datos en dispositivos locales. Además, las conexiones remotas suelen ser cifradas, lo que protege la información transmitida.

05. **Ahorro de costos**:
Reduce la necesidad de mantener hardware costoso y actualizado en cada estación de trabajo, ya que la potencia del procesamiento y almacenamiento reside en el servidor remoto.

- Ejemplos de herramientas para acceso y gestión de escritorios remotos:

TeamViewer:
Una herramienta popular que permite acceso remoto y soporte técnico a través de internet.

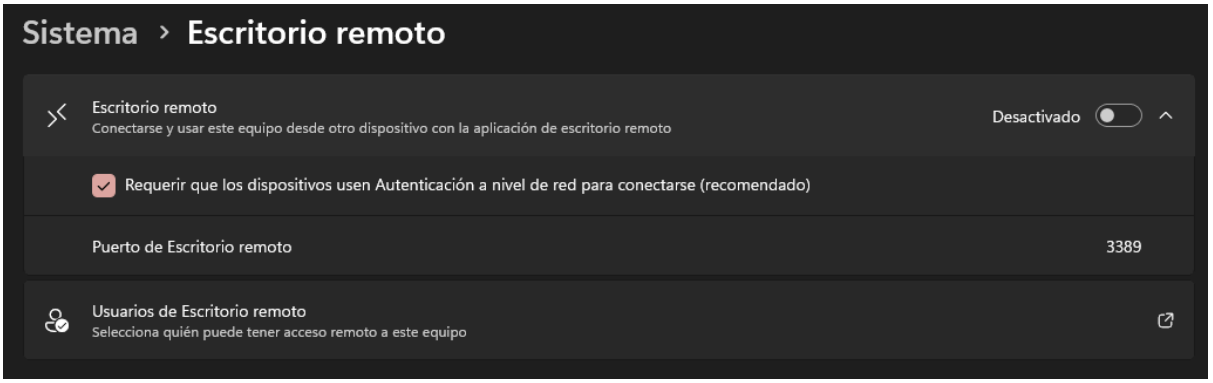
Remote Desktop Protocol (RDP):
En protocolo desarrollado por Microsoft que permite a los usuarios conectarse a un escritorio remoto en Windows.

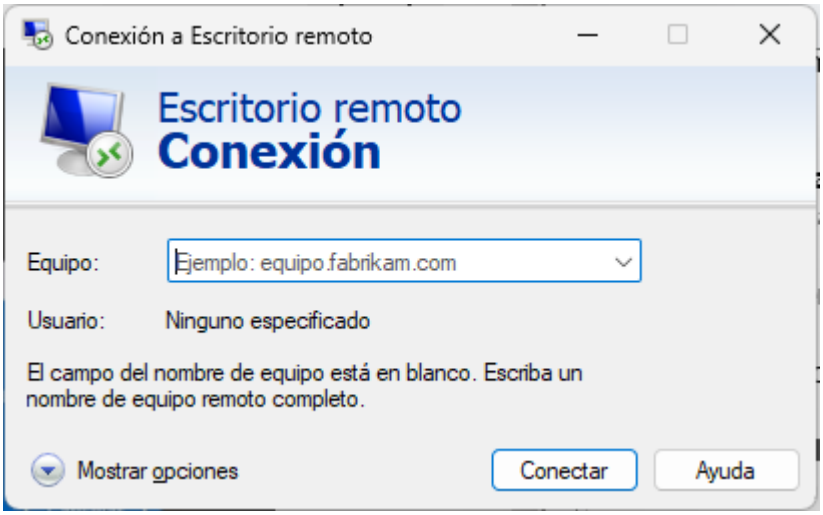
VNC (Virtual Network Computing):
Un estándar de código abierto que permite a los usuarios ver y controlar las computadoras remotas a través de una red.

Chrome Remote Desktop:
Una extensión de Google Chrome que permite a los usuarios acceder de forma remota a otro ordenador a través del navegador Chrome.

AnyDesk:
Similar a TeamViewer, AnyDesk permite el acceso remoto y la colaboración en línea entre múltiples usuarios.

1.5. Práctica Escritorio Remoto





ipconfig en terminal para ver la configuración de IP.

1.6. Máquinas Virtuales

- Máquinas virtuales:

Las máquinas virtuales (VM, por sus siglas en inglés) son entornos de software que simulan un sistema informático completo, incluyendo hardware, sistema operativo, y aplicaciones, dentro de un entorno aislado autónomo. Estas máquinas virtuales permiten ejecutar múltiples sistemas operativos y aplicaciones en un único servidor físico o computadora, lo que brinda una mayor flexibilidad y eficiencia en el uso de recursos.

- Importancia en el desarrollo y despliegue de software:

01. **Ambientes Aislados**:

Permiten crear entornos de desarrollo y prueba aislados, lo que facilita probar nuevas configuraciones de software sin afectar al entorno de producción.

02. **Portabilidad**:

Las máquinas virtuales son fácilmente transportables entre diferentes sistemas físicos, lo que simplifica el proceso de despliegue y migración de aplicaciones.

03. **Optimización de Recursos**:

Permiten utilizar eficientemente los recursos de hardware al compartir un servidor fisico entre múltiples máquinas virtuales, reduciendo costos y maximizando la utilización de recursos.

04. **Escalabilidad**:

Facilitan la escalabilidad del sistema, ya que es posible agregar o quitar máquinas virtuales según la necesidad de carga de trabajo.

05. **Respaldo y Recuperación**:

Proporcionan la capacidad de realizar respaldos completos de máquinas virtuales, lo que simplifica la recuperación en caso de fallos o desastres.

- Ventajas de Utilizar Máquinas Virtuales para de Desarrollo y Prueba de Software:

01. **Aislamiento y Seguridad**:

Las máquinas virtuales proporcionan un entorno aislado y seguro para realizar pruebas y experimentos, sin afectar al sistema operativo o aplicaciones del sistema anfitrión.

02. **Flexibilidad**:

Permiten crear y destruir entornos de desarrollo y prueba de manera rápida y sencilla, adaptándose a las necesidades del proyecto.

03. **Reproducibilidad**:

Facilitan la reproducción de escenarios de prueba y la depuración de errores al proporcionar entornos consistentes y controlados.

04. **Optimización de Recursos**:

Al compartir recursos de hardware entre múltiples máquinas virtuales, se optimiza el uso de recursos y se reduce el costo total de propiedad.

05. **Escalabilidad**:

Las máquinas virtuales pueden escalar vertical y horizontalmente según las necesidades de carga de trabajo, lo que permite adaptarse a cambios en la demanda de recursos.

1.7. Práctica Máquinas Virtuales

vmware workstation. Es un software de máquinas virtuales.

2. Checkpoints de contenidos

- ¿Cuál es el propósito principal de un lenguaje de programación?

Permitir que los programadores escriban instrucciones que las computadoras puedan ejecutar.

- ¿Cuál es la diferencia entre un compilador y un intérprete?

Un compilador traduce todo el código fuente a código máquina de una vez, mientras que un intérprete traduce y ejecuta el código línea por línea.

- ¿Qué característica distingue a un IDE de un simple editor de texto para programación?

Un IDE integra herramientas como depurador, compilador y gestor de código, que no están en los editores de texto básicos.

- ¿Cuál es una ventaja de usar escritorios remotos?

Permiten acceder y controlar un sistema informático desde cualquier lugar.

- ¿Para qué se utiliza comúnmente una máquina virtual en el desarrollo de software?

Para simular diferentes entornos operativos y probar software en ellos sin afectar al sistema en donde se ejecuta.