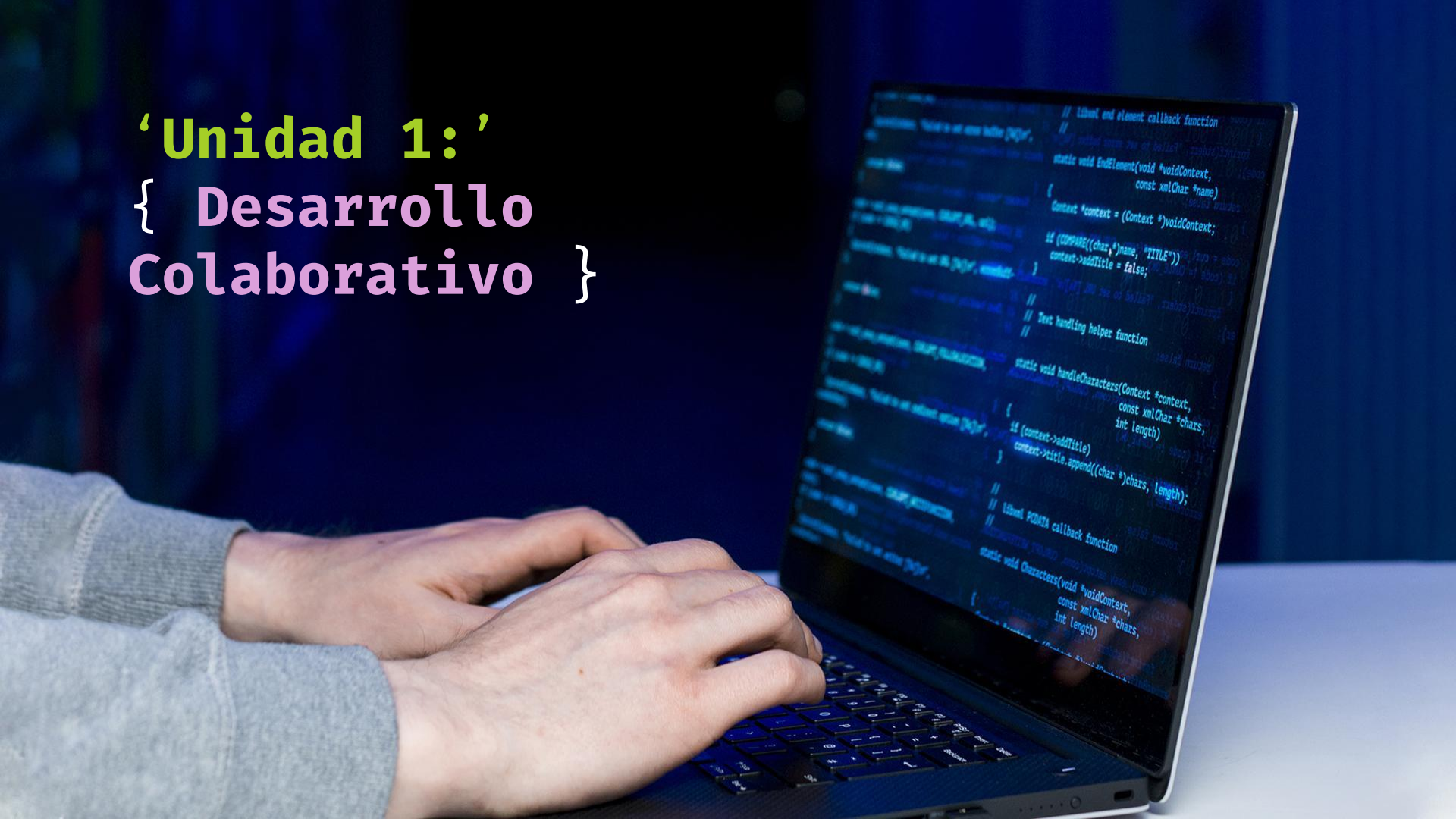

‘Unidad 1:’
{ Desarrollo
Colaborativo }

CONTENIDOS DE LA CLASE {

- Software de Control de Versiones
- Fundamentos de Git
- Gitflow
- Markdown

}

OBJETIVOS {

- Comprender qué son los sistemas de control de versiones y por qué son esenciales en el desarrollo de software. Conocer las alternativas actuales.
- Dominar los fundamentos de Git, desde su instalación y creación de proyectos hasta el manejo de ramas y comandos básicos.
- Entender el enfoque de Gitflow para gestionar proyectos, incluyendo la implementación de características, lanzamientos y correcciones de errores.
- Familiarizarse con Markdown y su uso en la documentación de repositorios.

}

```
{
}
```

[1. Software de Control de Versiones]

¿Qué son y para qué se utilizan?{

< El Software de Control de Versiones (SCV), también conocido como Sistema de Control de Versiones o VCS (Version Control System), es una herramienta que permite gestionar y rastrear cambios en archivos y proyectos de software a lo largo del tiempo.

Su principal objetivo es registrar todas las modificaciones realizadas en un conjunto de archivos, lo que facilita la colaboración, el seguimiento de cambios, la recuperación de versiones anteriores y la resolución de conflictos en equipos de desarrollo de software. >

}

Evolución del Software de Control de Versiones {

* <Sistema de Control de Versiones Local (VCS)

  < Los primeros sistemas de control de versiones eran locales, lo que significa que solo se podían utilizar en una sola máquina. Ejemplos de esto incluyen RCS (Revision Control System) y SCCS (Source Code Control System). >

  * <Sistema de Control de Versiones Centralizado (CVCS)>

    < Luego, se desarrollaron sistemas centralizados, como CVS (Concurrent Versions System) y Subversion (SVN), que permiten a varios desarrolladores trabajar en un mismo repositorio, pero con una única ubicación central para el código fuente. Aunque mejoraron la colaboración, aún tenían limitaciones. >

    * <Sistema de Control de Versiones Distribuido (DVCS)>

      < La evolución más reciente es la adopción de sistemas distribuidos como Git y Mercurial. Estos sistemas permiten que cada desarrollador tenga una copia completa del repositorio, lo que brinda mayor flexibilidad, rendimiento y resistencia a fallos. >

}

Alternativas actuales {

- [Git] < Es uno de los sistemas de control de versiones distribuidos más populares. Fue creado por Linus Torvalds y se utiliza ampliamente en el desarrollo de software de código abierto y en la industria. >
- [Subversion (SVN)] < Aunque menos popular que Git, SVN sigue siendo utilizado en algunas organizaciones. Es un sistema centralizado de control de versiones. >
- [Mercurial] < Similar a Git, Mercurial es otro sistema de control de versiones distribuido que se utiliza en una variedad de proyectos. >
- [Perforce] < Es una opción popular en la industria del desarrollo de videojuegos y en equipos que trabajan con activos binarios de gran tamaño. >

}

```
{
}
```

[2. Fundamentos de Git]

Caracteristicas de Git {

< Git es un sistema de control de versiones de código abierto diseñado para rastrear cambios en archivos y coordinar el trabajo entre múltiples personas.

Algunas de sus características clave incluyen: >

< Distribuido: En Git, cada colaborador tiene una copia completa del repositorio en su máquina local. Esto permite trabajar de manera descentralizada y sin conexión a Internet. >

< Historial de cambios: Git almacena un historial completo de cambios en cada archivo, lo que facilita la recuperación de versiones anteriores y la identificación de quién hizo cada cambio. >

< Ramas: Git permite crear ramas independientes en el repositorio para trabajar en nuevas características o correcciones de errores sin afectar la rama principal del proyecto. >

< Velocidad y eficiencia: Git es conocido por su velocidad y eficiencia en la gestión de proyectos, incluso en proyectos grandes. >

}

Ramas en Git{

< En Git, las ramas son líneas de desarrollo independientes que permiten a los equipos de desarrollo trabajar en varias características o solucionar problemas simultáneamente sin interferir con el trabajo de los demás. Cada repositorio Git tiene al menos una rama, que se llama generalmente "master" o "main", y representa la rama principal del proyecto. Las ramas son esenciales para el desarrollo colaborativo y la gestión de versiones en Git. >

}

Ventajas de utilizar ramas {

<VENTAJAS>

< Aislamiento de cambios: Al crear una nueva rama, se puede trabajar en una característica o solución de error específica sin afectar la rama principal. Esto evita conflictos entre colaboradores y mantiene la estabilidad del proyecto. >

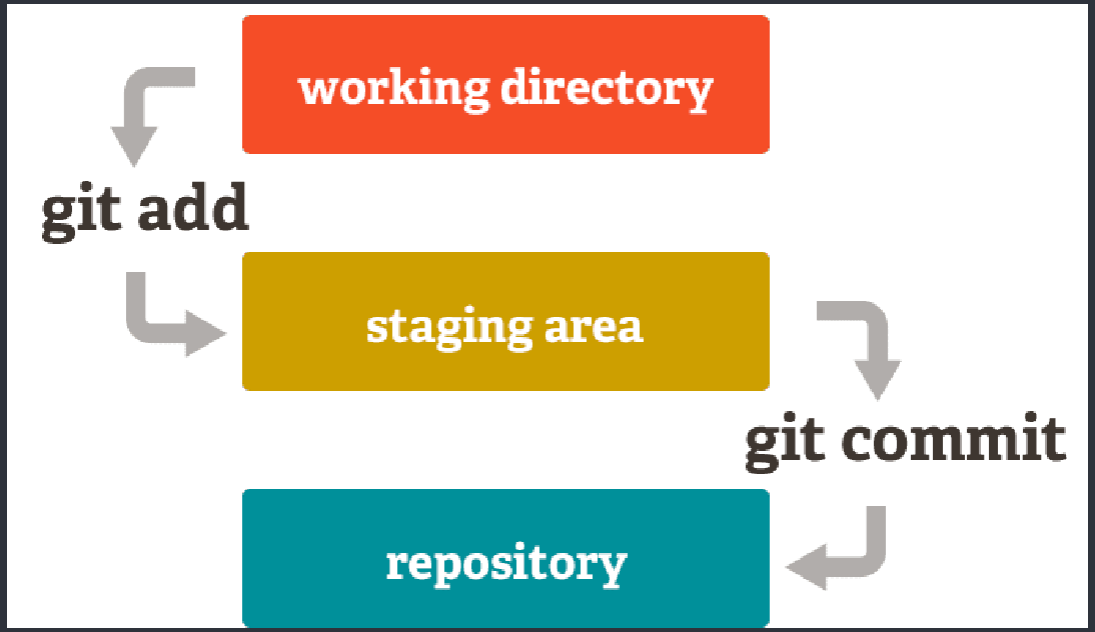
< Experimentación: Las ramas permiten experimentar con nuevas ideas sin comprometer el estado del proyecto principal. Si la idea no funciona, simplemente se puede eliminar la rama.>

< Paralelismo: Diferentes equipos o desarrolladores pueden trabajar en sus propias ramas, lo que acelera el desarrollo y permite realizar cambios simultáneos en diversas partes del proyecto.>

<Seguimiento de características: Cada rama puede utilizarse para desarrollar una característica específica. Esto facilita el seguimiento de las características implementadas y la identificación de quién está trabajando en qué parte del proyecto.>

}

Estados en Git {



}

Estados en Git: Working Tree (Árbol de Trabajo) {

< El Working Tree o Árbol de Trabajo en Git se refiere al directorio del proyecto en el que está trabajando activamente. Es el lugar donde se edita, crea y elimina archivos. En el Working Tree, los archivos son visibles y accesibles.

Los aspectos clave del Working Tree incluyen: >

< Archivos modificados: En esta área, se puede realizar cambios en los archivos existentes o crear nuevos archivos. Git no rastrea automáticamente estos cambios; se deben agregar explícitamente al área de preparación (Staging Area) para que se incluyan en el próximo commit. >
< Estado no rastreado: Los archivos que se encuentran en el Working Tree pero no se han agregado al área de preparación se consideran no rastreados. Git no los incluirá en los commits a menos que se los agreguen manualmente. >
< Acceso inmediato: Se tiene acceso completo a los archivos y se puede ver su contenido en su estado actual en el Working Tree. >

}

Estados en Git: Staging Area (Área de Preparación) {

< El Staging Area o Área de Preparación es un espacio intermedio entre el Working Tree y el Repository. Es donde se prepara y organiza los cambios que se desean incluir en el próximo commit. Es un componente fundamental de Git que proporciona un mayor control sobre los commits.

Las funciones del Staging Area son: >

< Selección de cambios: Se puede seleccionar y agregar archivos específicos o partes de archivos al Staging Area. Esto permite incluir solo los cambios que se desea en el próximo commit. >

< Revisión de cambios: Antes de confirmar los cambios en un commit, se pueden revisar en el Staging Area para asegurarse de que estén completos y correctos.

< Comentarios y descripciones: Se pueden agregar mensajes de commit que describan los cambios que se realizaron, lo que facilita la comprensión de los commits en el historial del proyecto. >

< Flexibilidad: Si se comete un error en el Working Tree, se puede realizar correcciones y volver a agregar los archivos al Staging Area antes de hacer un nuevo commit. >

}

Estados en Git: Repository (Repositorio) {

< El Repository o Repositorio es la base de datos donde Git almacena el historial completo de todos los commits realizados en el proyecto, incluidas las versiones anteriores de los archivos. Se encuentra en la carpeta .git en el directorio raíz del proyecto y es esencial para mantener el control de versiones y la historia del proyecto.

Las características del Repository incluyen: >

< Historial completo: El Repository almacena todas las versiones anteriores de los archivos, lo que permite acceder a cualquier estado anterior del proyecto. >

< Seguridad de datos: Los datos en el Repository están protegidos contra la pérdida y la corrupción, lo que garantiza la integridad del historial del proyecto. >

< Acceso colaborativo: El Repository se puede compartir con otros colaboradores, lo que facilita la colaboración y el seguimiento de cambios en el proyecto. >

< Revertir cambios: Si es necesario, se puede retroceder a versiones anteriores del proyecto para solucionar problemas o recuperar datos. >

}

```
{
}
```

[3. Gitflow]

¿Qué es Gitflow?{

< Gitflow es un conjunto de reglas y prácticas que se utilizan con Git para organizar el flujo de trabajo en proyectos de desarrollo de software. Fue desarrollado por Vincent Driessen y se basa en la gestión de ramas (branching) para facilitar la colaboración y la administración de versiones en proyectos. >

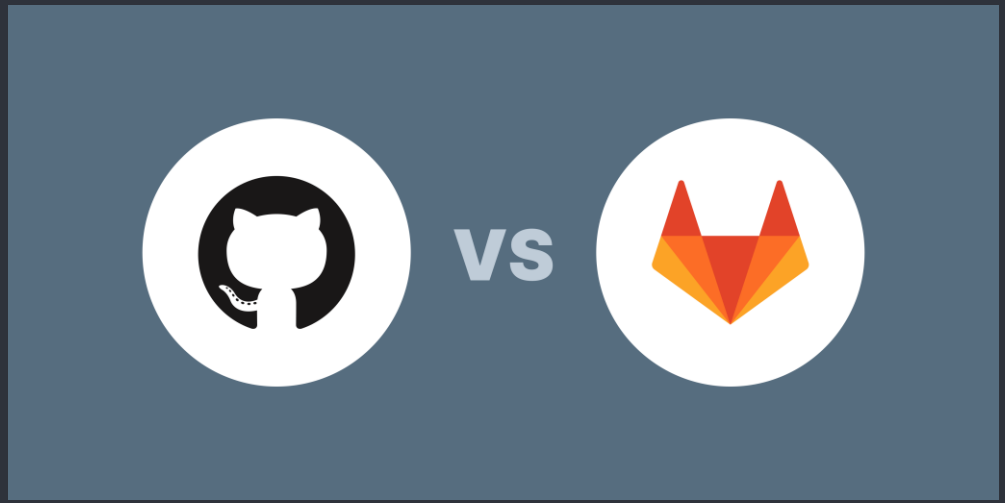
}

```
Características de Gitflow  {


    Ramas principales {
        <En Gitflow, generalmente existen dos ramas principales:
        "master" (o "main") y "develop". La rama "master" refleja
        el código en producción, mientras que la rama "develop" es
        donde se integran y prueban características nuevas. >
     }

    Ramas de características (feature branches) {
        < Cada nueva característica o mejora se desarrolla en una
        rama de características independiente. Estas ramas se crean
        a partir de "develop" y, una vez completadas, se fusionan
        nuevamente en "develop". >
     }
}
```

Características de Gitflow {

```
Ramas de lanzamiento (release branches) {
    <Antes de cada lanzamiento, se crea una rama de lanzamiento a
    partir de "develop". En esta rama, se realizan pruebas
    finales, corrección de errores y preparativos para el
    lanzamiento. Una vez listo, se fusiona con "master" y
    "develop". >
 }

Ramas de corrección de errores (hotfix branches) {

    < Si se descubre un error crítico en producción, se crea una
    rama de corrección de errores a partir de "master". Una vez
    corregido el error, se fusiona con "master" y "develop". >

 }
}
```

```
Software de Control de Versiones en la Nube {

}
```



```
Software de Control de Versiones en la Nube: GitHub {

    < GitHub es una de las plataformas más populares de
    alojamiento de repositorios de control de versiones
    en la nube, basada en Git. Ofrece una amplia gama de
    características que facilitan la colaboración, la
    gestión de proyectos y la automatización en el
    desarrollo de software.  >

}
```

Software de Control de Versiones en la Nube: GitHub {

<CARACTERISTICAS>

<**Repositorios**: GitHub permite alojar repositorios Git públicos y privados, lo que facilita la gestión de código fuente para proyectos de software. Los repositorios públicos son accesibles para todos, mientras que los privados requieren acceso autorizado. >

<Colaboración: La plataforma está diseñada para facilitar la colaboración en proyectos de desarrollo. Varias personas pueden contribuir al mismo repositorio, y GitHub ofrece herramientas para gestionar el acceso, revisar y aprobar solicitudes de cambio (pull requests) y realizar un seguimiento de las actividades.>

<Gestión de problemas: GitHub incluye un sistema de seguimiento de problemas que permite a los equipos registrar problemas, solicitar nuevas características, asignar tareas y seguir el progreso. Cada problema se puede discutir y etiquetar para una organización más eficiente.>

<Gestión de versiones y ramas: GitHub facilita el uso de Git y ofrece una interfaz web intuitiva para trabajar con ramas, realizar fusiones (merges) y rastrear el historial de cambios en el repositorio. Las funcionalidades de colaboración se integran de manera natural en este flujo de trabajo.>

}

Software de Control de Versiones en la Nube: GitLab {

< GitLab es otra plataforma de alojamiento de repositorios basada en Git que compite directamente con GitHub. Además de ofrecer alojamiento de repositorios, GitLab se destaca por su enfoque en DevOps y proporciona un conjunto integral de herramientas para el desarrollo de software y la administración del ciclo de vida de las aplicaciones: >

}

Software de Control de Versiones en la Nube: GitLab {

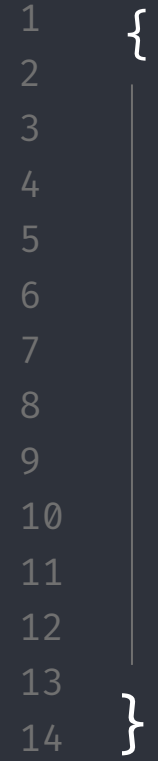
<CARACTERISTICAS>

<**Ciclo completo de DevOps:** GitLab incluye herramientas para la planificación de proyectos, seguimiento de problemas, control de versiones, integración continua, entrega continua y monitoreo de aplicaciones, todo en una sola plataforma. Esto permite a los equipos gestionar el ciclo de vida completo de una aplicación desde una sola interfaz. >

<Versiones comunitarias y empresariales: GitLab ofrece tanto versiones de código abierto (Community Edition) como versiones empresariales con características avanzadas (Enterprise Edition). Esto brinda opciones para equipos de desarrollo de todos los tamaños y necesidades.>

<Seguridad y cumplimiento: GitLab se enfoca en la seguridad y el cumplimiento normativo. Proporciona análisis de seguridad de código, escaneo de vulnerabilidades y características de cumplimiento para garantizar que el código se mantenga seguro y cumpla con las regulaciones.>

}



[4. Markdown]

¿Qué es Markdown? {

> < Markdown es un lenguaje de marcado ligero que se utiliza para formatear texto de una manera fácil de escribir y leer. Fue creado por John Gruber y Aaron Swartz en 2004 y se ha convertido en una de las herramientas más populares para crear documentación, blogs y contenido en línea. Markdown se utiliza comúnmente en sitios web, plataformas de blogging y sistemas de control de versiones, como GitHub y GitLab. >

}

Caracteristicas de Markdown {

- < Sintaxis sencilla: Markdown utiliza una sintaxis sencilla y legible que no requiere conocimientos técnicos avanzados para utilizarlo. >
- < Plataforma independiente: El formato Markdown se puede usar en una variedad de aplicaciones y plataformas, lo que lo hace altamente portable. >
- < Soporte amplio: La mayoría de las plataformas de blogging, sistemas de control de versiones y editores de texto admiten Markdown de manera nativa o a través de extensiones. >
- < Fácil de aprender: Aprender Markdown no lleva mucho tiempo, lo que lo hace accesible para una amplia audiencia, desde desarrolladores hasta escritores. >
- < Personalizable: Markdown es extensible y se puede personalizar con extensiones para adaptarse a las necesidades específicas. >

}

Documentación del repositorio con lenguaje de marcado ligero: {

La documentación del repositorio con un lenguaje de marcado ligero como Markdown es una práctica común en el desarrollo de software. La forma en la que se utiliza Markdown para documentar repositorios es creando un archivo README.md. Utilizarlo en una plataforma como GitHub o GitLab, tiene múltiples ventajas: {

<Ejemplos de uso: Markdown es especialmente útil para proporcionar ejemplos de código y comandos en la documentación. Se puede usar bloques de código formateados para indicar como hacer funcionar el proyecto, lo que hace que la documentación sea valiosa para los desarrolladores y usuarios. >

<Colaboración: El uso de Markdown en la documentación facilita la colaboración. Varios colaboradores pueden contribuir al documento, revisar y corregir errores, y mantenerlo actualizado con facilidad. >

}

}