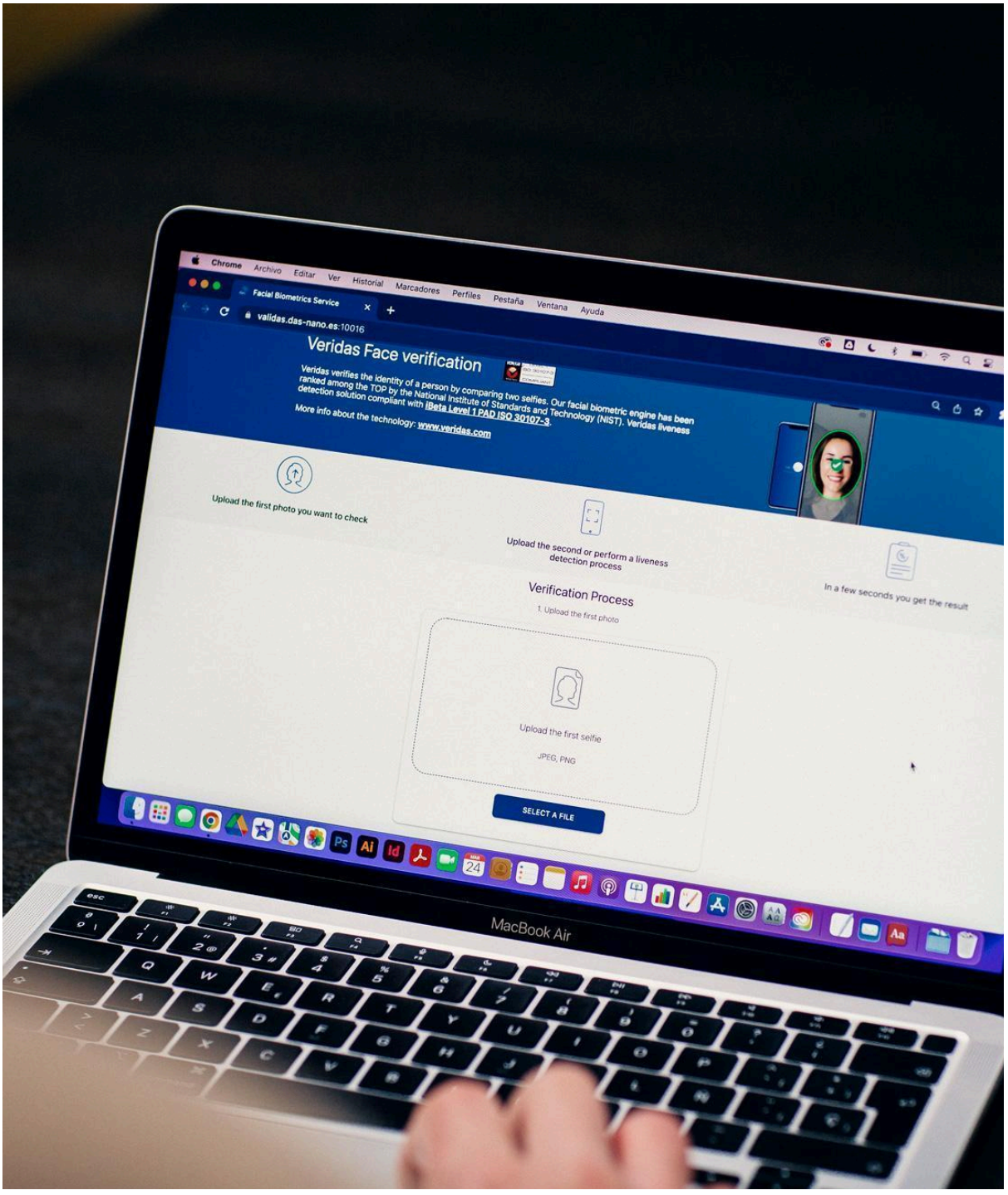
ALBAÑILESDIGITALES
Developing your new career

servicio navarro de empleo

das-nano veridas



Módulo 0 – INTRODUCCIÓN AL DESARROLLO DE SOFTWARE

Objetivos y motivaciones

Este módulo inicial es consecuencia de que en ediciones anteriores se observó que los alumnos tenían dificultades para el seguimiento del curso debido a una carencia de conocimientos básicos. Debido a ello, con este módulo:

- Se pretenden impartir estos conocimientos para un buen desarrollo del curso por parte de los alumnos.
- Se pretende igualar el nivel de conocimientos base de los alumnos.

Fundamentos del desarrollo de Software y herramientas (7h)

Introducción al mundo del desarrollo de Software (1/2h)

¿Qué es programar?

Ejercicio de preguntar a todos qué es programar. Finalmente: programar es decirle a una máquina que haga lo que tú quieres.

Patas del desarrollo de Software

Programar está relacionado con el mundo del desarrollo de software, que no es lo mismo. Podríamos decir que en el desarrollo de software se compone distintas patas: **diseño**, **implementación**, **operaciones**, **monitorización** y **mantenimiento**. **Programar sería la segunda parte**. Comparación con la construcción de una casa:

Diseño:

- En el desarrollo de software, el diseño implica definir los requisitos y especificaciones de la aplicación. Los equipos de desarrollo trabajan en estrecha colaboración con los clientes o stakeholders para comprender sus necesidades y expectativas.
- Al igual que en la construcción de una casa, los arquitectos y diseñadores crean planos y bocetos detallados para la casa, teniendo en cuenta las preferencias del cliente y los aspectos funcionales.

Implementación:

- La implementación se refiere a la fase en la que los desarrolladores comienzan a construir la aplicación utilizando lenguajes de programación y herramientas adecuadas. Aquí es donde se traducen los diseños en código funcional.
- En la construcción de una casa, la implementación sería el proceso en el que los constructores utilizan materiales y herramientas para construir la estructura física según los planos y especificaciones.

Operaciones:

- En esta etapa, la aplicación web está lista para su funcionamiento en un entorno de producción. Se configura la infraestructura necesaria para alojar y ejecutar la aplicación.
- De manera análoga, en una casa, las operaciones serían la fase en la que se instalan los servicios públicos, como electricidad, agua, y gas, para que la casa sea funcional y habitable.

Monitorización:

- En el desarrollo de software, la monitorización implica supervisar el rendimiento y el funcionamiento de la aplicación en tiempo real. Esto permite detectar problemas y tomar medidas correctivas si es necesario.
- En una casa, la monitorización podría compararse con el mantenimiento periódico y la revisión de sistemas y estructuras para asegurarse de que todo está en buen estado y funcionando correctamente.

Mantenimiento:

- La fase de mantenimiento en el desarrollo de software consiste en corregir errores, actualizar características y realizar mejoras en la aplicación con el tiempo.
- En la construcción de una casa, el mantenimiento se refiere a la reparación de posibles daños o el reemplazo de elementos que se han desgastado con el uso.

Breve repaso a lenguajes de programación y sus usos

Si programar es decirle a una máquina que haga lo que quieres, lenguajes de programación hay muchos, igual que lenguajes con los que nos comunicamos los humanos. Cada lenguaje suele tener un nicho concreto, algo en lo que es especialmente bueno.

Ejemplo con vehículos que se usan para distintos propósitos. Coches para el día a día, motos para ir por el campo o la montaña, camiones para transportar mercancías, barcos para ir sobre el agua, o vehículos que te sirvan para un gran número de situaciones.
De la misma manera, encontraremos:

- **Lenguajes de alto nivel**: versátiles, se adaptan a distintas situaciones. Por ejemplo, Python.
- **Lenguajes de propósito específico**: especializados en una tarea. Para transportar 10 toneladas usas un camión, no una bicicleta.
- **Lenguajes de bajo nivel**: C, C++. Mucho control o cercanía al hardware, pero son más difíciles de aprender.

Al igual que en el desarrollo de software, donde cada lenguaje tiene su propósito específico y ventajas, en el mundo del transporte, cada tipo de vehículo es adecuado para diferentes escenarios y necesidades de movilidad. De manera similar, los desarrolladores pueden elegir el lenguaje de programación más adecuado según el tipo de proyecto y los objetivos específicos que deseen lograr.

Introducción a Unix, Linux, y concepto de open-source (1/2h)

Unix: Unix es uno de los primeros sistemas operativos desarrollados en la década de 1960 en los Laboratorios Bell de AT&T. Fue diseñado como un sistema multiusuario y multitarea, con una arquitectura modular que permite a los desarrolladores escribir programas pequeños y especializados que realizan tareas específicas. Unix ha sido una base importante para muchos otros sistemas operativos y ha tenido un impacto significativo en el desarrollo de software.
Linux: Linux es un sistema operativo que fue desarrollado por Linus Torvalds en 1991. A diferencia de Unix, Linux es de código abierto, lo que significa que su código fuente es accesible y puede ser modificado y distribuido por cualquier persona bajo ciertas licencias de software libre. Esto ha llevado a que Linux se convierta en una plataforma popular para la colaboración y desarrollo comunitario. Linux está basado en los principios y la arquitectura de Unix, pero no es un Unix propiamente dicho. Es más bien una reimplementación del sistema operativo Unix de código abierto y compatible con los estándares de Unix.

Open source (Wikipedia): El código abierto es un modelo de desarrollo de software basado en la colaboración abierta. Se enfoca en los beneficios prácticos (acceso al código fuente) y en cuestiones éticas o de libertad que tanto se destacan en el software libre. Para muchos el término «libre» hace referencia al hecho de adquirir un software de manera gratuita. Sin embargo, de lo que se trata es de abaratar los costos y ampliar la participación; que sea libre no necesariamente implica que sea gratuito, lo importante sigue siendo ampliar la participación y extender libertades.

Beneficios del software open source

1. Libertad de ver, modificar y distribuir el código fuente del software.
2. Colaboración entre desarrolladores: COMUNIDAD.
3. La mejora continua del software y la reducción de costos.
4. Rapidez a la hora de desarrollar: ejemplo con la inteligencia artificial.

Recursos

- https://www.youtube.com/watch?v=UUJ0dFpj1-M
- https://www.youtube.com/watch?v=NDhJfHhe3e4

Navegando con el terminal (4h)

¿Qué es el terminal?

Comenzaremos definiendo qué es una **interfaz**: es la forma en la que distintos elementos o componentes interactúan, se comunican, o se relacionan entre sí. Algunas interfaces están pensadas para que los humanos interactuemos con el software, como las interfaces de usuario, o **UI**. Pero hay otras interfaces con las que podemos interactuar de forma distinta con cualquier elemento.
El **terminal** no es más que otra interfaz, otra forma de interaccionar con el ordenador. Puedes hacer lo mismo que con la UI a través de comandos. Es muy potente y veréis que lo usaréis mucho.

Conceptos importantes del terminal

- **Comandos**: los comandos son instrucciones que se utilizan en el terminal para realizar diversas tareas, como navegar por el sistema de archivos,

crear, copiar o eliminar archivos, y ejecutar programas. Cada comando tiene una función específica y se ingresa en la línea de comandos para que el sistema lo ejecute.

- **Argumentos**: los argumentos son información adicional que se proporciona junto con un comando para especificar qué acción debe realizar el comando. Los argumentos pueden ser nombres de archivos o directorios en los que se aplicará el comando, opciones que afecten el comportamiento del comando o cualquier otra información necesaria para completar la tarea.
- **Directorios**: los directorios son carpetas que se utilizan para organizar y almacenar archivos en el sistema de archivos. Los directorios pueden contener otros directorios y archivos, lo que crea una estructura jerárquica que facilita la organización de datos en el sistema.
- **Rutas**: las rutas son ubicaciones específicas en el sistema de archivos que se utilizan para referenciar archivos y directorios. Hay dos tipos de rutas:
    - Rutas **absolutas**: son rutas que especifican la ubicación completa del archivo o directorio desde la raíz del sistema de archivos. Comienzan con una barra "/" en sistemas basados en Unix (por ejemplo, /home/usuario/archivo.txt).
    - Rutas **relativas**: son rutas que especifican la ubicación de un archivo o directorio en relación con la ubicación actual en el sistema de archivos. No comienzan con una barra "/" y pueden usar referencias como "." (directorio actual) o ".." (directorio padre) (por ejemplo, ../documentos/archivo.txt).
- **Opciones**: las opciones son modificadores que se pueden agregar a un comando para personalizar su comportamiento. Generalmente, se especifican utilizando guiones (por ejemplo, "-a" o "--verbose"). Las opciones pueden activar funciones adicionales o cambiar el resultado del comando.

Navegación por el sistema de archivos

Comandos: pwd, ls, mkdir, touch, mv, rm

Demostración.

Visualización y edición de archivos

Comandos: cat, head, tail, nano, vim
Demostración.

Operaciones y redirección

1. **Pipes** (|): El operador de tubería (|) se utiliza para redirigir la salida estándar de un comando hacia la entrada estándar de otro comando. Esto permite que los resultados de un comando se utilicen como entrada para otro.
2. **Redireccionamiento de Salida** (>): El operador de redireccionamiento de salida (>) se utiliza para redirigir la salida estándar de un comando hacia un archivo. Si el archivo ya existe, se sobrescribirá.
3. **Redireccionamiento de Salida** (>>): El operador de redireccionamiento de salida (>>) se utiliza para redirigir la salida estándar de un comando hacia un archivo. Si el archivo ya existe, el nuevo contenido se agregará al final del archivo.
4. **Operadores Lógicos** (&& y ||): Los operadores lógicos && (AND) y || (OR) se utilizan para ejecutar comandos en función de si el comando anterior tuvo éxito (&&) o falló (||).

Ejercicios prácticos

Realizar los ejercicios del archivo *ejercicios_terminal.pdf*

Recursos

- https://github.com/jlevy/the-art-of-command-line
- https://ubuntu.com/tutorials/command-line-for-beginners#1-overview

Introducción a IDEs (2h)

Qué es un IDE y qué importancia tiene en el desarrollo de Software

Un **IDE** (Entorno de Desarrollo Integrado) es una aplicación de software que proporciona un conjunto de herramientas y funcionalidades integradas para

facilitar el desarrollo de software. Reúne diversas herramientas como un editor de código, un depurador, un compilador, un sistema de control de versiones y más, en una única interfaz.

El desarrollo de software y los IDEs se pueden explicar utilizando la analogía de un cocinero y su cocina. Al igual que un cocinero utiliza su cocina como el espacio de trabajo principal para preparar deliciosos platillos, los desarrolladores utilizan los IDEs como su entorno de desarrollo integral para escribir, editar y depurar código de software. Así como un cocinero tiene diferentes utensilios y herramientas en su cocina para cortar, mezclar y cocinar ingredientes, los IDEs ofrecen un conjunto de herramientas integradas, como editores de código, depuradores y sistemas de control de versiones, para facilitar el proceso de desarrollo. La cocina proporciona el espacio organizado donde un cocinero puede crear y experimentar con sabores, mientras que los IDEs ofrecen un entorno estructurado para los desarrolladores donde pueden trabajar en proyectos, gestionar archivos y colaborar con otros miembros del equipo.

Ventajas de usar un IDE:

- **Productividad**: Los IDEs ofrecen características como el resaltado de sintaxis, completado de código, plantillas y funciones inteligentes, lo que acelera el proceso de desarrollo.
- **Depuración**: Los IDEs permiten depurar código paso a paso y encontrar errores con facilidad, lo que facilita la identificación y corrección de problemas.
- **Gestión de Proyectos**: Los IDEs ofrecen una organización eficiente de proyectos, lo que permite trabajar en múltiples archivos y directorios sin complicaciones.
- **Integración con Herramientas Externas**: Los IDEs pueden integrarse con sistemas de control de versiones, pruebas unitarias y otras herramientas externas, mejorando la colaboración y el flujo de trabajo.

Instalación y familiarización con VS Code

VSCode es un popular IDE de código abierto desarrollado por Microsoft. Es altamente personalizable, ligero y extensible, lo que lo convierte en una elección popular para desarrolladores de diversas tecnologías.

Instalación: seguir la documentación oficial:
https://code.visualstudio.com/docs/setup/linux#_installation

Una vez lo tenga todo el mundo instalado:

- Explicar extensiones útiles, como linters, formateadores de código, etc.
- Explicar interfaz de usuario: que incluye la barra de menú, la barra de herramientas, la barra lateral y el área de edición de código.
- Crear un proyecto, agregar archivos, ver estructura en barra lateral.
- Editar, escribir, copiar, pegar o eliminar código.
- Personalización: preferencias y temas.

Recursos

- https://www.youtube.com/watch?v=Ei1y51K8jQk excepto secciones "Control de versiones con git" y "Debugger, el depurador de VSCode".

Conceptos básicos sobre el desarrollo de Software (5h)

Cliente - Servidor (1/2h)

Para explicar esta sección contaremos con dos voluntarios que simularán ser un panadero y una persona que va a comprar el pan, de la forma más educada posible, desde que entra en la tienda hasta que se marcha. El objetivo es ver que en una comunicación entre un cliente y un servidor hay una figura que pide algo, un consumidor, y otra figura que provee un recurso, un servidor. Del mismo modo, hay que observar que para que eso sea posible, no se puede entrar como un elefante en una cacharrería, tiene que haber un protocolo de comunicación para poder entenderse y que la comunicación sea fluida.

Modelo cliente servidor

Cliente: la parte de la aplicación que solicita y consume servicios o recursos del servidor.
Servidor: la parte de la aplicación que proporciona y gestiona los servicios o recursos solicitados por el cliente.
Se comunican a través de un protocolo, que no hace falta que conozcan en este momento.

Papel de clientes y servidores en aplicaciones de Software

Poner el ejemplo de **Instagram**: cuando te metes en la aplicación de tu móvil, no tienes las fotos de la gente que sigues, le pides al servidor que te la dé, este verifica lo que haga falta, y si procede, te la devuelve.
Más ejemplos: sistema de correo electrónico, videojuegos online, o cualquier red social.

Control de versiones (2h)

Es un sistema que registra y gestiona los cambios realizados en el código fuente y otros archivos a lo largo del tiempo. El control de versiones es crucial en el desarrollo de software, destacando la colaboración, el seguimiento de cambios y la posibilidad de revertir a versiones anteriores.
Ejemplo:
Imagina que estás trabajando en un proyecto de grupo con tus compañeros de clase para escribir una historia. Cada miembro del grupo es responsable de escribir diferentes partes de la historia. A medida que colaboran, te das cuenta de que llevar un registro de los cambios y coordinar el trabajo de todos se está volviendo un poco complicado. Aquí es donde entra en juego el control de versiones:

- **Sin Control de Versiones**: en ausencia de control de versiones, cada miembro del grupo podría escribir su propia parte de la historia por separado y luego enviar sus versiones a una persona que intenta fusionar manualmente todos los cambios. Se vuelve difícil rastrear quién escribió qué, cuándo se hicieron los cambios y cómo evolucionaron las diferentes versiones de la historia con el tiempo. Además, si alguien borra o modifica accidentalmente una parte de la historia, podría ser difícil restaurarla a un estado anterior.
- **Con Control de Versiones**: usando el control de versiones, podrías utilizar una plataforma digital como Git para escribir la historia de manera colaborativa.

Github y cómo vamos a usarlo

Realizar el trabajo del archivo *github.pdf*

Docker (2,5h)

Introducción a contenedores y Docker

Para explicar qué es docker, haremos un símil entre el desarrollo de software y la cocina. Imagina que eres un chef que prepara deliciosos platillos en tu cocina personal. Quieres llevar tus creaciones a diferentes lugares: a tu restaurante, a eventos, o incluso a la casa de un amigo para una cena especial. ¿Qué problemas encontrarías? Cada lugar tiene su propia cocina con diferentes utensilios, ingredientes y temperaturas de cocción. Esto puede llevar a inconsistencias y dificultades en la preparación.

Cocina personal (entorno de desarrollo): en tu propia cocina, tienes todo lo que necesitas para cocinar tus platillos: ingredientes, herramientas, recetas y un flujo de trabajo que conoces bien. Aquí es donde desarrollas y pruebas tus recetas con éxito.

Restaurante (entorno de producción): cuando intentas llevar tus platillos a un evento o tu restaurante, te enfrentas a desafíos. Las cocinas son diferentes, los utensilios varían y las temperaturas de cocción pueden no ser las mismas. Esto puede afectar la calidad de tus platillos y llevar a resultados impredecibles.

Solución con Docker (mini-cocinita): imagina que tienes una mini-cocinita especial en la que empacas todos los ingredientes, utensilios y recetas exactos que necesitas. Cada vez que llevas tus platillos a diferentes lugares, simplemente sacas tu mini-cocinita y tienes todo lo que necesitas para cocinar de manera consistente.

Beneficios:

- **Replicación del entorno**: al igual que tu mini-cocinita siempre contiene los mismos elementos, un contenedor Docker siempre contiene los mismos componentes y configuraciones. Esto asegura que la aplicación se comporte de la misma manera en cualquier lugar.
- **Facilidad de despliegue**: llevar tu mini-cocinita a diferentes lugares es tan simple como sacarla y comenzar a cocinar. De manera similar, desplegar un contenedor Docker en diferentes entornos es sencillo y predecible.

Los contenedores son la tecnología que permite empaquetar instancias de tus aplicaciones, y docker es la tecnología sobre los contenedores que lo permite.

Ejemplos prácticos de docker y docker-compose

Realizar los ejercicios del archivo *docker_guide.pdf*

Recursos

- https://www.youtube.com/watch?v=4Dko5W96WHg

Competencias esenciales en desarrolladores de Software (1h)

Inglés

1. **Documentación y Recursos Técnicos**: Gran parte de la documentación, tutoriales, guías y recursos técnicos están disponibles en inglés. Comprender y acceder a estos recursos es esencial para aprender nuevas tecnologías y resolver problemas.
2. **Lenguaje Universal de la Tecnología**: El inglés es ampliamente utilizado como el lenguaje de comunicación en la industria tecnológica a nivel global. Colaborar con equipos internacionales y participar en proyectos de código abierto a menudo requiere comunicarse en inglés.
3. **Herramientas y Plataformas**: Muchas herramientas, bibliotecas y plataformas utilizan interfaces en inglés. Saber cómo navegar y utilizar estas herramientas es crucial para el desarrollo efectivo de software.
4. **Comunicación con Compañeros**: El inglés es el medio común de comunicación entre desarrolladores de diferentes regiones y culturas. Compartir ideas, discutir problemas y colaborar en proyectos se vuelve más fluido si todos pueden comunicarse en un idioma común.
5. **Comunicación con Clientes**: En un entorno empresarial, es posible que tengas que interactuar con clientes, ya sea para obtener requisitos, proporcionar actualizaciones o resolver problemas. El inglés facilita estas comunicaciones, especialmente si tus clientes son de diferentes países.
6. **Participación en la Comunidad Tecnológica**: Participar en conferencias, eventos y foros internacionales es enriquecedor para tu crecimiento

profesional. Muchos de estos eventos se llevan a cabo en inglés, y poder interactuar con otros profesionales en el mismo idioma es fundamental.

Seguir buenas prácticas

1. **Calidad y Mantenibilidad del Código**: Las buenas prácticas promueven la creación de código limpio, legible y estructurado. Esto facilita la comprensión del código por parte de otros desarrolladores y permite realizar cambios y mejoras de manera más eficiente en el futuro.
2. **Reducción de Errores y Fallos**: Las buenas prácticas incluyen la validación y prueba exhaustiva del código. Esto ayuda a identificar y corregir errores antes de que lleguen a producción, lo que disminuye la posibilidad de fallos y problemas inesperados.
3. **Colaboración Efectiva**: Cuando los equipos siguen buenas prácticas, se establece un estándar común que facilita la colaboración. Los desarrolladores pueden trabajar en diferentes partes del código y entender cómo interactúan entre sí.
4. **Eficiencia en el Desarrollo**: Las buenas prácticas incluyen el uso de patrones de diseño, reutilización de código y desarrollo modular. Esto acelera el proceso de desarrollo al evitar la reinvención constante y promover la eficiencia en la escritura de código.
5. **Seguridad y Cumplimiento**: Las buenas prácticas de seguridad en el desarrollo ayudan a prevenir vulnerabilidades y ataques cibernéticos. Siguiendo prácticas como la validación de entradas y el manejo seguro de datos, se minimizan los riesgos de seguridad.
6. **Adaptabilidad y Escalabilidad**: Siguiendo buenas prácticas, el código está diseñado de manera que sea más fácil agregar nuevas funcionalidades y escalar el sistema. Esto es crucial a medida que las aplicaciones crecen y evolucionan con el tiempo.
7. **Facilita el Aprendizaje**: Las buenas prácticas proporcionan pautas claras para desarrolladores nuevos y experimentados. Esto acelera el proceso de aprendizaje y reduce la curva de adaptación al trabajar en proyectos existentes.
8. **Reputación Profesional**: Los desarrolladores que siguen buenas prácticas tienden a ganar reputación por su habilidad para crear software de alta

calidad. Esto puede abrir oportunidades laborales y aumentar la confianza de los empleadores y colaboradores.

Ejemplo: consumir y producir buena documentación. Las buenas prácticas incluyen la capacidad de comprender y aplicar bibliotecas, frameworks y herramientas externas. Consumir la documentación proporcionada por los creadores de estas tecnologías es esencial para utilizarlas de manera efectiva y aprovechar todas sus características. Por otro lado, documentar tu propio código y proyectos es clave para comunicar su funcionamiento, propósito y uso a otros desarrolladores. Proporcionar una documentación clara y concisa facilita la colaboración y permite que el proyecto sea mantenido y mejorado con éxito en el futuro.

Técnicas eficaces de búsqueda por Internet

La habilidad de buscar online es esencial para resolver problemas, aprender y mantenerse actualizado en el desarrollo de software. **Google** es una herramienta poderosa para encontrar información relevante y soluciones a problemas técnicos. Hay que saber usarlo.
Demostración con **matching operators**:

- desarrollo de software -> desarrollo de software -ibm (o lo primero que salga, para eliminarlo)
- desarrollo de software buenos dias -> desarrollo de software “buenos dias” (contiene lo entrecomillado)
- desarrollo de software -> desarrollo de software site:.edu (buscar por dominio)
- desarrollo de software -> desarrollo de software filetype:pdf (por tipo de archivo)

Github y Stach Overflow
Son dos sitios de referencia en los que habitualmente es buena idea mirar y consumir recursos. Están fuertemente ligados a la comunidad, que se ayude y resuelve dudas de forma altruista. Aun así, siempre es necesario verificar y adaptar las soluciones encontradas para que se ajusten a las necesidades y especificaciones del proyecto.

Recursos

- https://www.youtube.com/watch?v=BRiNw490Eq0 -> curso de búsqueda en Google.

Mantenerse actualizado

Importancia del aprendizaje continuo en tecnología

Mantenerse actualizado como desarrollador de software es crucial por varias razones fundamentales:

1. **Rápida Evolución Tecnológica**: la industria tecnológica avanza a un ritmo excepcionalmente rápido. Nuevas tecnologías, lenguajes de programación, marcos de trabajo y herramientas emergen constantemente. Mantenerse al día garantiza que estés al tanto de las últimas tendencias y puedas aplicar las soluciones más actuales a los problemas.
2. **Calidad del Trabajo**: la actualización constante asegura que puedas abordar desafíos con soluciones modernas y eficientes. Esto se traduce en productos y aplicaciones de mayor calidad, más robustos y que cumplen con los estándares actuales.
3. **Competitividad en el Mercado Laboral**: en un mercado laboral competitivo, los empleadores valoran a los desarrolladores que pueden aportar habilidades frescas y relevantes. Mantenerse al día con las últimas tecnologías y prácticas te hace más atractivo para oportunidades laborales y avances en tu carrera.
4. **Mejora Continua de Habilidades**: a medida que aprendes nuevas habilidades y tecnologías, también mejoras tus habilidades existentes. Esto te hace un desarrollador más versátil y capaz de abordar una variedad de proyectos y desafíos.
5. **Resolución eficaz de Problemas**: mantenerse actualizado te proporciona un conjunto más amplio de herramientas para resolver problemas. Puedes aplicar soluciones innovadoras a desafíos recurrentes y evitar quedar atrapado en métodos obsoletos.

Cómo mantenerse actualizado

1. **Lectura Constante**: dedicar tiempo a leer blogs técnicos, artículos y noticias relevantes en la industria. Esto te mantiene al tanto de las últimas tendencias, herramientas y tecnologías emergentes.
2. **Seguir a Referentes**: seguir a expertos en desarrollo de software en redes sociales, como Twitter y LinkedIn. Sus publicaciones y comparticiones pueden proporcionar información valiosa y enlaces a recursos útiles.
3. **Participación en Comunidades**: unirte a comunidades en línea, foros y grupos de discusión relacionados con el desarrollo de software. Estos espacios ofrecen oportunidades para aprender de otros y discutir temas actuales.
4. **Asistencia a Eventos Técnicos**: asistir a conferencias, meetups y seminarios sobre desarrollo de software. Estos eventos proporcionan la oportunidad de escuchar a expertos, aprender sobre nuevas tecnologías y establecer contactos.

Oratoria

La oratoria es fundamental en cualquier situación laboral, y concretamente en el desarrollo de software es importante porque es habitual encontrarte con cualquiera de las siguientes situaciones:

1. **Comunicación con Equipos y Clientes**: los desarrolladores de software a menudo trabajan en equipos multidisciplinarios y deben comunicarse efectivamente con otros desarrolladores, diseñadores, gerentes y clientes. La habilidad de expresarse claramente y transmitir ideas de manera coherente es esencial para colaborar de manera efectiva.
2. **Explicación de Conceptos Técnicos**: en la industria de la tecnología, hay conceptos complejos y técnicos que deben explicarse a personas no técnicas, como clientes, gerentes de proyectos o usuarios finales. La oratoria permite traducir términos técnicos en lenguaje comprensible.
3. **Presentaciones y Reuniones**: los desarrolladores pueden necesitar presentar sus proyectos, actualizaciones y hallazgos en reuniones, conferencias o demostraciones. Una buena oratoria garantiza que puedan comunicar sus ideas de manera clara y cautivadora.

4. **Negociación y Persuasión**: en situaciones como negociaciones contractuales o discusiones sobre enfoques de desarrollo, la habilidad de persuadir y negociar es fundamental. Una comunicación efectiva permite llegar a acuerdos mutuamente beneficiosos.
5. **Gestión de Problemas**: en momentos complejos o cuando surgen problemas técnicos, los desarrolladores deben comunicar rápidamente la situación, las acciones tomadas y los próximos pasos. La oratoria clara evita malentendidos y fomenta la colaboración en la resolución de problemas.

Conceptos avanzados del desarrollo de Software (3h)

Introducción a la Inteligencia Artificial

Principios técnicos básicos

François Chollet (creador de Keras): la **inteligencia** es la eficiencia en la adquisición de habilidades nuevas. Para poder medir la inteligencia hay que tener en cuenta: los conocimientos y experiencias previas, dificultad de generalización y alcance.
Marvin Minski (científico estadounidense, considerado uno de los padres de la IA): la **inteligencia artificial** es la ciencia de hacer máquinas capaces de hacer tareas que requerirían inteligencia si las hicieran los humanos.
Podemos decir que los humanos adquirimos conocimientos de tres formas principales:

1. **Deducción**: se trata de ir de lo general a lo particular. Ejemplo: los mamíferos son animales, los perros son mamíferos, los perros son animales.
2. **Inducción**: se trata de observar patrones a través de la repetición. Ejemplo: veo que el sol sale todos los días por la mañana, pues aprendo que el sol cada día sale por la mañana.
3. **Abducción**: se centra en encontrar la mejor explicación posible para un conjunto de datos o hechos. Ejemplo: si veo muchos charcos en la calle,

puede ser que alguien haya regado todo el suelo, pero lo más probable es que haya llovido.

Las máquinas son mucho mejores que los humanos en la inducción. Es decir, a través de la repetición, al proporcionarle una cantidad ingente de información, es capaz de observar patrones que los humanos somos incapaces.

Aplicaciones

- **Asistentes virtuales y chatbots**: las IA como Siri, Google Assistant y chatbots en sitios web brindan asistencia y respuestas a preguntas de manera interactiva.
- **Procesamiento del lenguaje natural**: las aplicaciones de NLP permiten la traducción automática, análisis de sentimientos, resumen de texto, generación de contenido y más.
- **Visión por computadora**: las tecnologías de visión por computadora permiten la detección de objetos, reconocimiento facial, análisis de imágenes médicas, vehículos autónomos y más.
- **Salud y medicina**: la IA se utiliza para el diagnóstico médico, la detección temprana de enfermedades, la personalización de tratamientos y la investigación de nuevos medicamentos.
- **Finanzas**: en finanzas, la IA se aplica para el análisis de riesgos, detección de fraudes, predicción de mercados y gestión de carteras.
- **Automatización industrial**: la IA se utiliza en la automatización de procesos de fabricación, control de calidad y mantenimiento predictivo.
- **Robótica**: la robótica avanzada, incluidos los robots colaborativos y los robots autónomos, se benefician de la IA para realizar tareas complejas y adaptarse a entornos cambiantes.
- **Educación**: la educación personalizada y el aprendizaje adaptativo utilizan la IA para ofrecer contenidos y enfoques de aprendizaje personalizados.
- **Juegos**: la IA se ha utilizado en la creación de oponentes virtuales desafiantes y en el diseño de estrategias en juegos.
- **Agricultura**: la agricultura de precisión utiliza sensores y análisis de datos impulsados por IA para optimizar el rendimiento de los cultivos y el uso de recursos.

- **Automatización de procesos empresariales**: la IA se aplica para automatizar tareas repetitivas en empresas, mejorando la eficiencia y reduciendo errores.
- **Búsqueda en la web y recomendaciones**: los motores de búsqueda y los sistemas de recomendación utilizan IA para proporcionar resultados y sugerencias relevantes.
- **Transporte**: la IA se utiliza en vehículos autónomos y en la optimización de rutas de entrega y transporte público.
- **Energía**: la gestión de redes eléctricas, la optimización de la producción de energía y la eficiencia energética son áreas donde la IA puede tener un impacto significativo.
- **Investigación científica**: la IA ayuda en el análisis de datos científicos complejos, como en la investigación genómica y en la simulación de fenómenos naturales.
- **Influencia en sectores de la sociedad, elecciones**: recomendar el capítulo del podcast "La tertulia" donde hablan de ello.

Impacto en el desarrollo de Software

La IA tiene el potencial de tener un impacto significativo en el desarrollo de software de varias formas, como automatización de tareas, optimización de código, generación de código, detección de vulnerabilidades y seguridad, testing... de todas estas, son especialmente útiles dos a día de hoy: modelos de lenguaje capaces de generar código, como **ChatGPT**, y modelos propios de generación y optimización de código, como **Copilot**.

ChatGPT

Enseñar para qué uso ChatGPT y hacer ejercicios del archivo *ejercicios_IA.pdf*

Hugging Face y LM Studio

Hugging Face es una plataforma donde la comunidad del machine learning colabora con modelos, datasets y aplicaciones. Sirve también como repositorio donde se alojan modelos de IA que pueden ser usados por todo el mundo.
LM Studio es una aplicación con interfaz gráfica que te facilita la interacción con modelos de IA. Los modelos los saca, precisamente, de Hugging Face.

Hora de jugar con estas dos herramientas.