Recibe una cálida:

¡Bienvenida!

Te estábamos esperando 😁

alkemy

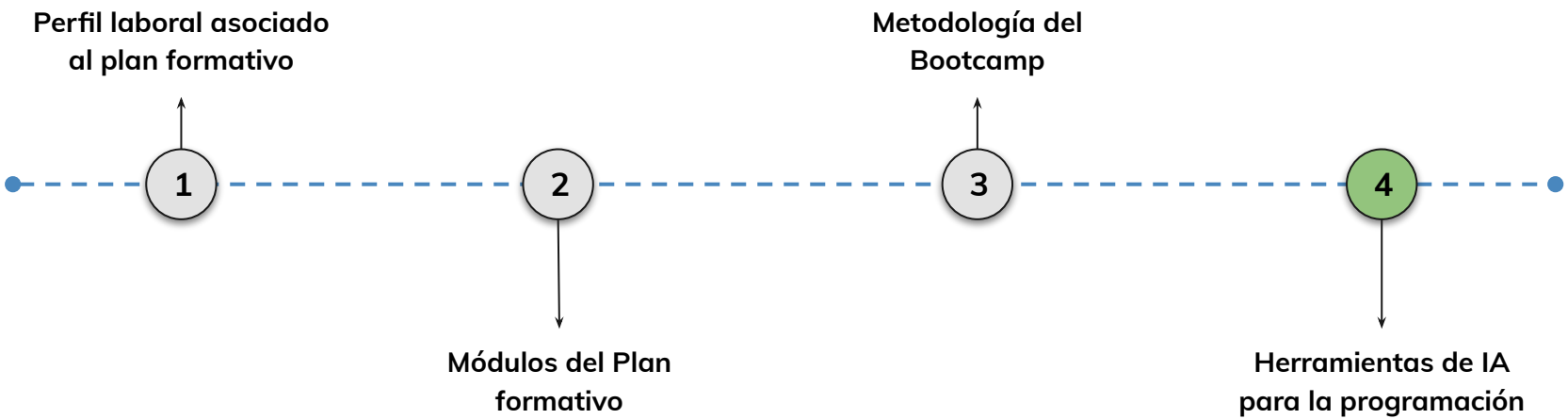
Clase grabada

› Herramientas de IA para la programación

Aprendizaje Esperado: utilizar herramientas de inteligencia artificial para el aumento de la productividad en la programación según las buenas prácticas de la industria.

Roadmap de lecciones

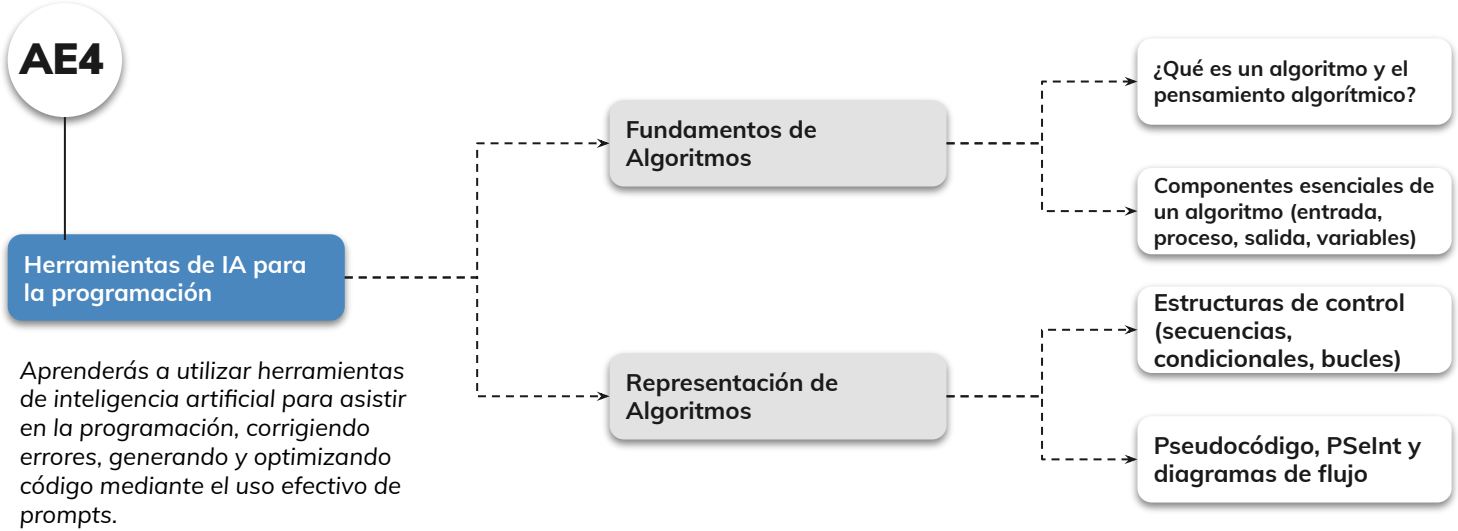
*¿Cuáles **lecciones** estaremos estudiando en este módulo?*



alkemy

Learning Path

¿Cuáles temas trabajaremos hoy?



alkemy

Objetivos de aprendizaje

¿Qué aprenderás?

- Comprender la estructura lógica y los elementos de un algoritmo.
- Representar algoritmos usando pseudocódigo claro y estructurado.
- Aplicar estructuras de control para resolver problemas reales.
- Usar PSeInt para simular y analizar algoritmos paso a paso.
- Construir diagramas de flujo que reflejen el proceso algorítmico.
- Identificar tipos de IAs disponibles y su propósito.
- Comprender cómo “propmtear” para refinar procesos y código.

Algoritmos y pseudocódigo

alkemy

¿Qué es un Algoritmo y el Pensamiento Algorítmico?

1 Concepto de Algoritmo

Un algoritmo es un conjunto finito de pasos ordenados y definidos para resolver un problema. Debe tener entrada, proceso, salida, determinismo y finitud.

2 Pensamiento Algorítmico

Habilidad lógica para resolver problemas creando algoritmos, descomponiendo tareas complejas en pasos manejables y previendo resultados.

3 Características Clave

Incluye secuencialidad, repetición, condicionalidad, abstracción y modularidad, fundamentales para una lógica estructurada.

Componentes esenciales de un Algoritmo

Datos de Entrada y Salida

La información que el algoritmo necesita para funcionar y los resultados generados tras el procesamiento.

Procesamiento

Conjunto de pasos, reglas o instrucciones lógicas que manipulan los datos.

Variables

Espacios de almacenamiento con nombre que contienen valores que pueden cambiar durante la ejecución del algoritmo (ej. números, texto, booleanos).

< > Estructuras de Control en Algoritmos

Secuencia

Las instrucciones se ejecutan en el orden en que aparecen.

Condicionales

Permiten ejecutar diferentes acciones según una condición (Si...Entonces...Sino).

Repetitivas

Permiten repetir un bloque de instrucciones varias veces (Mientras, Repetir-Hasta, Para).

El dominio de las variables y estructuras de control es crucial para representar cualquier proceso lógico, adaptar algoritmos a situaciones dinámicas y generar soluciones eficientes y reutilizables. Esto facilita la transición del pseudocódigo a lenguajes de programación reales.

< > Pseudocódigo para Representar Algoritmos

¿Qué es el Pseudocódigo?

Es una forma intermedia entre el lenguaje natural y la programación real, utilizada para describir la lógica de un algoritmo sin depender de la sintaxis de un lenguaje específico. No se ejecuta en una computadora, pero es fácilmente entendible por personas.

Propósito y Ventajas

Representa ideas de forma estructurada, describe la lógica sin errores de sintaxis, sirve como puente al código real y facilita el trabajo colaborativo. Es neutral, ideal para la enseñanza y una herramienta de planificación esencial.

< > Estructura y reglas del Pseudocódigo

Inicio y Fin	Delimitan el algoritmo.
Palabras Clave	Leer, Mostrar, Si, Entonces, Mientras, Para, etc.
Asignaciones	Uso de ← o =.
Sangrías	Reflejan bloques de código.
Claridad y Precisión	Evitar ambigüedades y usar nombres descriptivos.

Dominar el pseudocódigo es clave para organizar tus ideas antes de codificar. Si tu lógica está bien estructurada en pseudocódigo, convertirla en un programa funcional será mucho más fácil, independientemente del lenguaje de programación.

< > Utilización de PSeInt para Algoritmos

¿Qué es PSeInt?

Herramienta educativa para desarrollar lógica algorítmica con pseudocódigo estructurado. Permite escribir, ejecutar y depurar algoritmos sin un lenguaje formal.

Características Principales

Escribe pseudocódigo en español/inglés, ejecuta paso a paso, muestra estado de variables, incluye editor de diagramas de flujo y ofrece plantillas.

Instalación y Uso

Descarga desde pseint.sourceforge.net, sigue el asistente, abre la aplicación y comienza un nuevo algoritmo. Escribe, guarda (.psc) y ejecuta.

< > Utilización de PSeInt para algoritmos

Ejecución paso a paso y variables en PSeInt

Simulación Línea por Línea

Permite observar cómo se evalúan condiciones, ejecutan bucles y modifican variables, identificando errores lógicos y de inicialización.

1

Uso en PSeInt

Escribe el algoritmo, haz clic en "Paso a paso". Una ventana mostrará la instrucción actual, el estado de las variables y la consola de salida.

2

Utilidad Clave

Desarrolla intuición algorítmica, detecta errores invisibles como bucles infinitos y mejora la traducción entre pseudocódigo y código real.

3

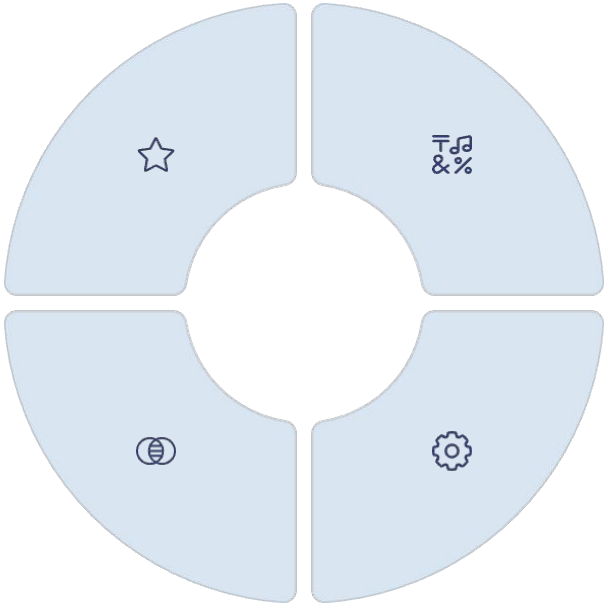
< > Diagramas de Flujo para Algoritmos

Representación Gráfica

Es un esquema visual que describe la secuencia lógica de pasos para resolver un problema, usando símbolos estandarizados y flechas.

Beneficios

Fomentan el pensamiento lógico, documentan algoritmos, ayudan a detectar errores y son excelentes para la enseñanza.



Símbolos Clave

Óvalos (inicio/fin), rectángulos (proceso), rombos (decisión), flechas (flujo) y paralelogramos (entrada/salida).

Herramientas

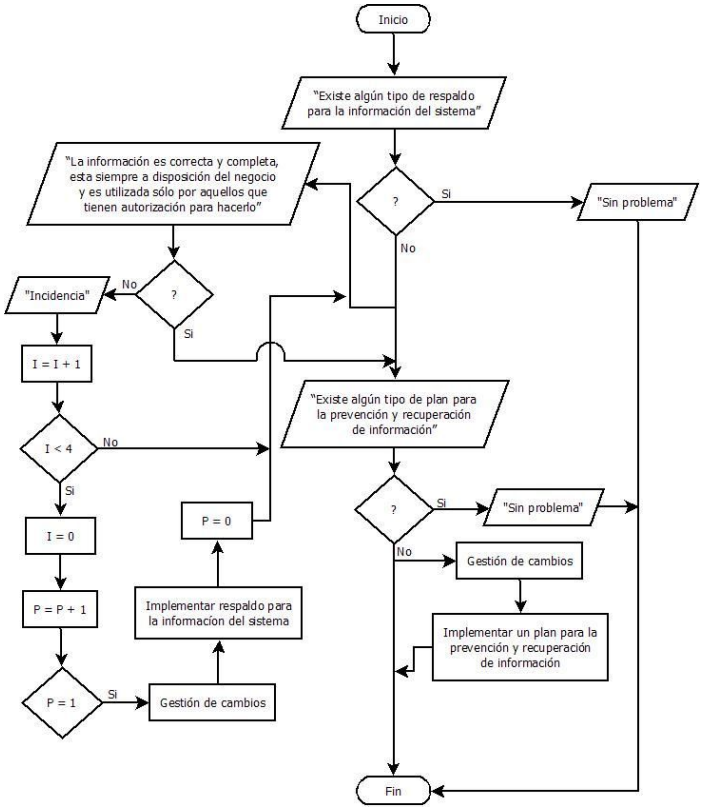
draw.io, Lucidchart, Pencil Project, yEd, y el propio PSeInt permiten crear y visualizar diagramas de flujo.

< > Complementando Algoritmos:

Pseudocódigo y Diagramas de Flujo

Ambas herramientas son fundamentales para el diseño y comprensión de algoritmos, ofreciendo perspectivas diferentes pero complementarias para su desarrollo.

Mientras que el pseudocódigo es ideal para la construcción detallada y la traducción directa a código, los diagramas de flujo brillan en la visualización del proceso y la identificación de la lógica global. Utilizar ambos mejora significativamente la comprensión y depuración de algoritmos.



alkemy

Pseudocódigo

Complementando algoritmos

Naturaleza textual: Permite una representación detallada de la lógica algorítmica, similar a un lenguaje de programación real.

Facilidad de escritura: Ideal para algoritmos complejos, donde la verbosidad y el detalle son necesarios sin preocuparse por la sintaxis estricta.

Transición a código: Sirve como un puente directo entre el pensamiento lógico y la codificación en un lenguaje de programación específico.

Énfasis en la estructura: Facilita la aplicación de estructuras de control (secuencia, decisión, repetición) de manera clara y organizada.

Diagramas de Flujo

Complementando Algoritmos

Representación gráfica: Ofrece una vista visual y clara del flujo de ejecución del algoritmo, ideal para la comprensión rápida.

Identificación de problemas: Ayuda a visualizar la secuencia de pasos y a detectar bucles infinitos, bifurcaciones incorrectas o faltas de conexión lógicas.

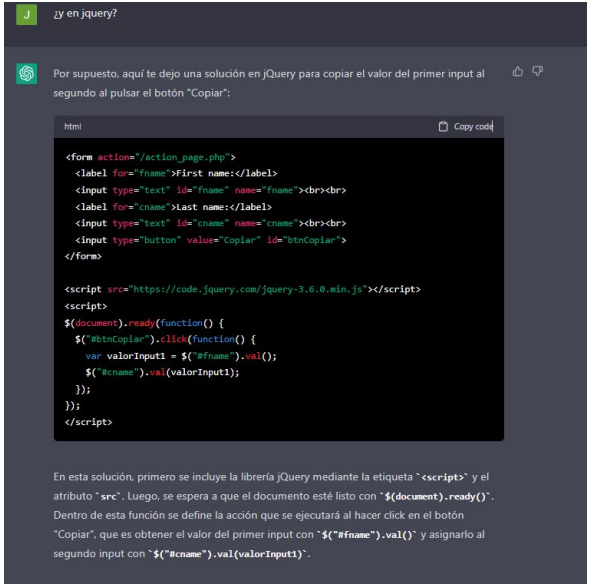
Comunicación efectiva: Excelentes para explicar algoritmos a personas con diferentes niveles de conocimiento técnico.

Enfoque en el proceso: Permite seguir el "camino" que toma un algoritmo bajo diferentes condiciones de entrada.

Herramientas de IA

< > Herramientas de IA para la productividad en la programación 🤖

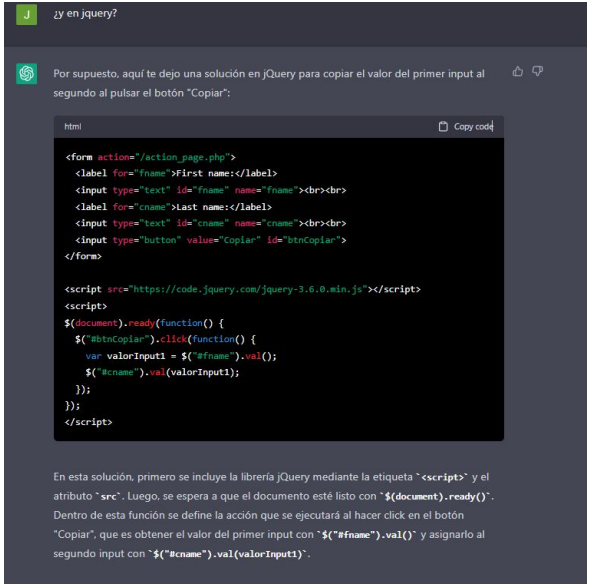
El desarrollo de software ha evolucionado más allá del código escrito manualmente. Las herramientas de Inteligencia Artificial están revolucionando la productividad del programador al **asistir en la escritura, corrección, generación, documentación y optimización del código.** Estas herramientas contribuyen en la reducción del timing en desarrollo permitiendo a su vez aumentar la calidad del software entregado.



alkemy

< > Herramientas de IA para la productividad en la programación 🤖

Con el auge de modelos de lenguaje como ChatGPT o asistentes como GitHub Copilot, el enfoque del programador moderno pasa a ser más estratégico: se trata de **saber qué pedir, cómo evaluar las respuestas de la IA**, **y cómo integrarlas con criterio** en su flujo de trabajo.



alkemy

< > IA en programación

Aplicaciones comunes de la IA en el desarrollo

A continuación, se detallan algunos casos concretos de aplicación de la IA en el desarrollo de software:

- **Asistencia en tiempo real**: mientras escribes código, la IA predice posibles líneas de continuación o funciones completas.
- **Autocompletado inteligente:** con sugerencias contextualizadas que entienden el código existente.
- **Explicación de código:** ideal para aprender lenguajes nuevos o comprender código legado.
- **Detección y corrección de errores:** análisis semántico y sintáctico del código con sugerencias claras.

< > IA en programación

Aplicaciones comunes de la IA en el desarrollo

- **Generación de código boilerplate:** por ejemplo, plantillas de clases o controladores en frameworks web.
- **Conversión entre lenguajes:** útil en migraciones o aprendizaje entre entornos.
- **Creación de pruebas unitarias:** generación automática de tests a partir del código fuente.

< > **ChatGPT** como asistente de programación

ChatGPT es uno de los asistentes conversacionales más potentes actualmente, con capacidad para interactuar con múltiples lenguajes de programación y ofrecer resultados útiles para tareas como:

- ✓ Consultas de sintaxis
- ✓ Corrección de errores de compilación o lógica
- ✓ Sugerencias de estructuras algorítmicas
- ✓ Revisión de seguridad en código
- ✓ Generación de documentación técnica (README, etc.)

Ejemplo:

Prompt: “Explícame qué hace este fragmento de código en [lenguaje de programación] y cómo puedo optimizarlo.”Resultado: Descripción detallada del propósito, análisis de eficiencia y sugerencias.

< > Diseño de **Prompts**

¿Qué hay que considerar al momento de analizar un prompt? 🤔

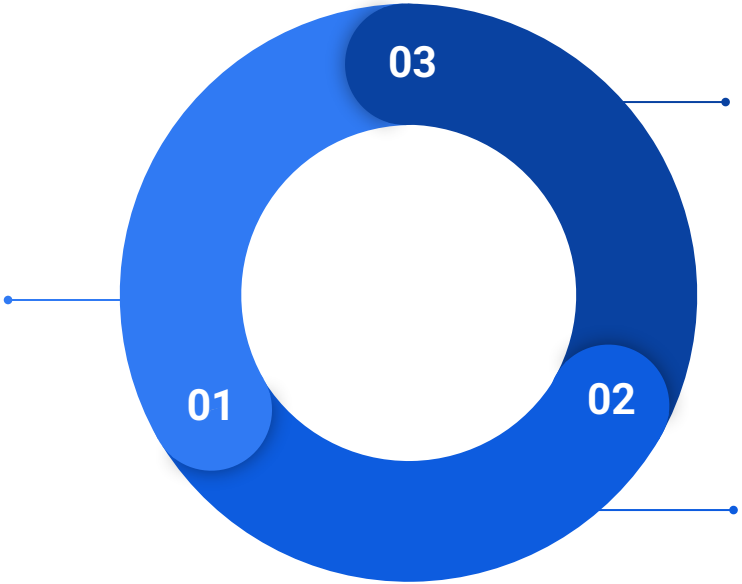
1. **Identifica el propósito del prompt:** ¿Qué necesitas obtener como respuesta?
2. **Escribe la acción:** Especifica qué debe hacer el modelo en términos claros.
3. **Define el contexto:** Proporciona la información esencial para enmarcar el problema.
4. **Añade detalles relevantes:** Proporciona requisitos o limitaciones para guiar la respuesta.

Estructura ideal de un **Prompt**

A considerar

Sé claro/a

Evita ambigüedades y términos generales.



Sé directo/a

Enfócate en lo que necesitas y evita sobrecargar el prompt con información innecesaria.

Sé específico/a

Proporciona detalles que permitan obtener una respuesta útil.

alkemy

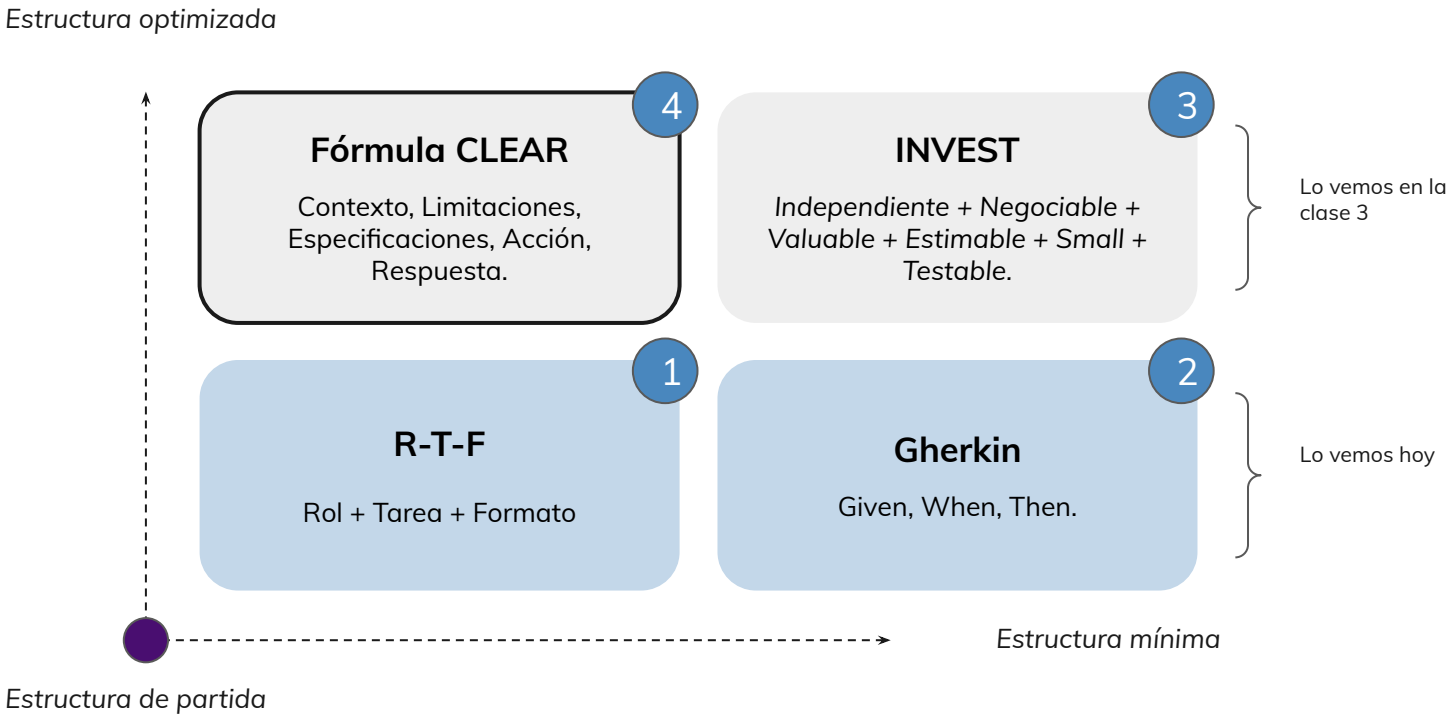
Momento:

Time-out!

⏰5 -10 min.

Modelos para estructurar Prompts

Modelos para estructurar Prompts



alkemy

RTF **Prompts**

Modelos para estructurar Prompts

Este modelo asegura que la IA actúe en un contexto claro y entregue respuestas alineadas a las expectativas. Es práctico y fácil de implementar en cualquier disciplina.

Fórmula:

Define el **Rol** de la IA + la **Tarea** específica + el **Formato** esperado de la respuesta.

- Actúa como [ROL]
- Crea una [TAREA]
- Muestra como [FORMATO]

Prompt inicial: "Actúa como un especialista en pruebas de software. Escribe un caso de prueba para validar un formulario de inicio de sesión. Proporciona el resultado en formato de tabla."

***Resultado esperado*:** Un caso de prueba organizado con columnas como *Caso de Prueba*, *Entrada Esperada*, y *Resultado Esperado*.

Gherkin **Prompts**

Modelos para estructurar Prompts

Este modelo, originado en Behavior-Driven Development (BDD), estructura casos de prueba y escenarios en un formato legible para humanos y máquinas, ideal para la automatización.

Fórmula:

Usa las palabras clave **Given** (Dado), **When** (Cuando), **Then** (Entonces).

- Provee el contexto para el escenario descrito [DADO]
- Especifica el conjunto de pasos a seguir [CUANDO]
- Específica el resultado esperado [ENTONCES]

Prompt inicial: "Dado que necesito probar la API de gestión de usuarios, cuando envíe una solicitud GET al endpoint '/users' con un token de autorización válido, entonces espero recibir un código de estado HTTP 200 y una lista de usuarios en formato JSON."

Resultado esperado: La IA podría generar un script en el lenguaje solicitado. Por ejemplo, en Python con requests.

INVEST **Prompts**

Modelos para estructurar Prompts

Originalmente creado para Historias de Usuario en metodologías ágiles, este modelo ayuda a desglosar las necesidades de un prompt en componentes claros, pequeños y accionables.

Disciplina: Agile (Scrum, User Story Development).

Fórmula:

- Independiente + Negociable + Valuable + Estimable + Small + Testable.

Prompt inicial: "Genera un script en [lenguaje] que valide el código de respuesta HTTP de una API REST. El script debe ser independiente, capaz de aceptar como entrada la URL y los headers opcionales, y devolver un resultado claro con un mensaje de éxito o error. Asegúrate de que sea modular y reusable, documentado con comentarios claros y fácil de integrar en un flujo de pruebas automatizadas."

Resultado esperado: Un script que es modular y puede integrarse fácilmente en flujos de automatización más amplios.

Fórmula CLEAR de **Prompt**

Fórmula recomendada

Esto suma en la medida que hacemos más claro el mensaje y nuestra necesidad

Multiplica tu resultado, indicando cómo quieres que se te muestre lo solicitado.

((Acción + Contexto + Dolor + Restricciones) / Herramienta específica) * Formato esperado

La división refiere a que todo lo que describiste en el Prompt debe estar atravesado por alguna herramienta para que el resultado sea el esperado

alkemy

Fórmula CLEAR de **Prompt**

Fórmula recomendada

Describe brevemente el caso de uso o el entorno.
"un script para probar un formulario de registro."

Agrega cualquier limitación técnica o funcional.
"sin usar librerías externas."

Especifica cómo quieres los resultados.
"en un archivo JSON con comentarios."

((Acción + Contexto + Dolor + Restricciones) / Herramienta específica) * Formato esperado

¿Qué quieres que haga la herramienta?
"Genera," "Valida," "Escribe," "Optimiza."

Detalla el objetivo específico.
"que valide campos obligatorios y formatos de correo electrónico."

Indica si hay una tecnología o framework particular.
"usando [Framework] en [Lenguaje de programación]."

alkemy

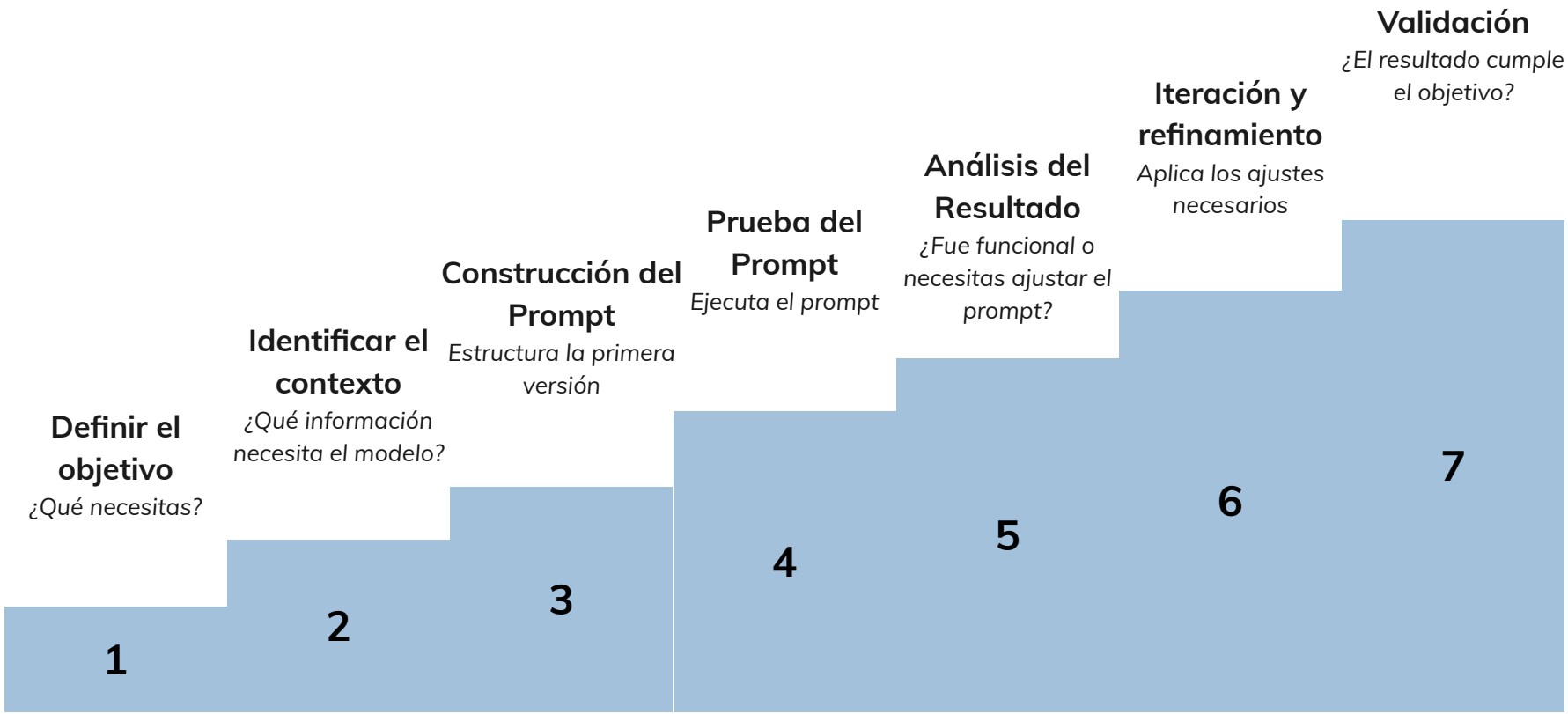
Framework de creación de Prompts

Algunos puntos a considerar

alkemy

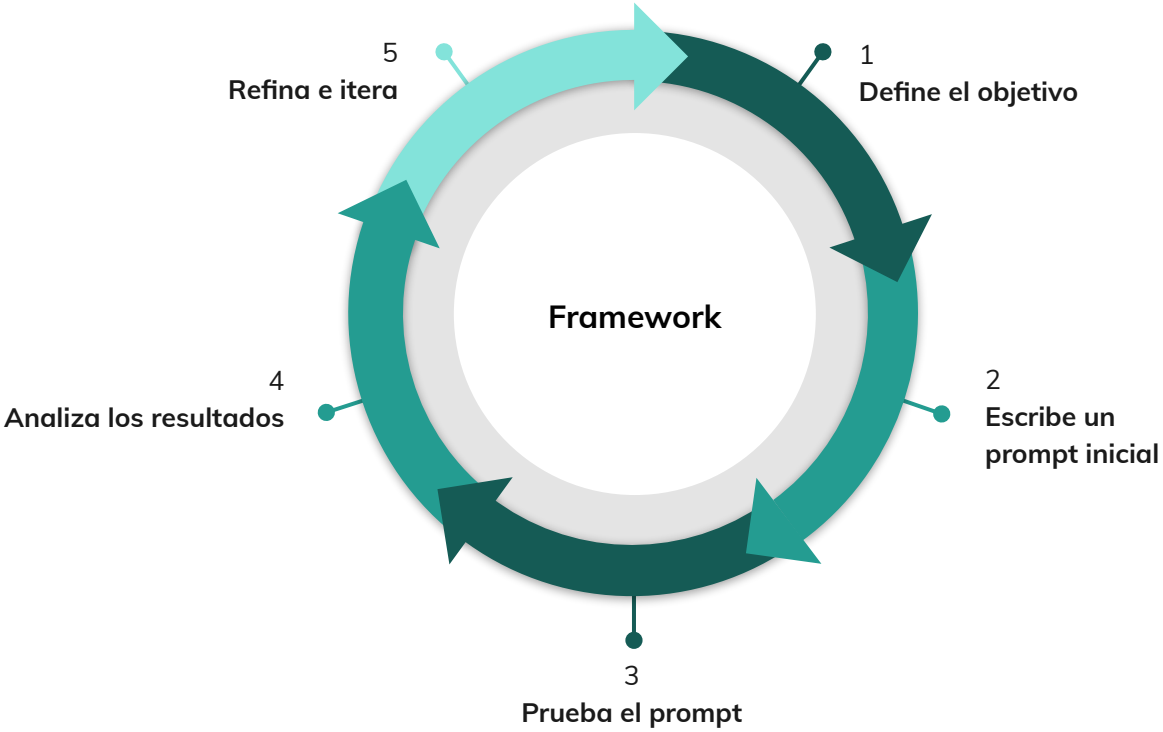
Framework de creación de Prompts

Ciclo de Ingeniería de Prompts



Framework de creación de Prompts

Ciclo de Ingeniería de Prompts



alkemy

Herramientas IA para Prompting 

Tipos de ***Inteligencia Artificial***

IA Generativa

Crea contenido original como texto, imágenes, música o código a partir de datos de entrada.

ChatGPT, DALL·E, Ollama.

IA Predictiva

Azure Machine Learning, Amazon Forecast.

Analiza datos históricos para prever resultados futuros.

IA Analítica

Procesa grandes volúmenes de datos para extraer patrones, insights o tendencias.

Google BigQuery, Tableau con IA.

IA Conversacional

ChatGPT, Google Assistant.

Interpreta y responde en lenguaje natural, simulando una conversación humana.

IA Adaptativa

Se ajusta dinámicamente a nuevos datos o entornos, mejorando su rendimiento sin intervención manual.

Hugging Face (ajustes de modelos), Dynamic Bayesian Networks

alkemy

Herramientas de ***Inteligencia Artificial***

Herramienta	Uso en productos digitales	Impacto / bondades
ChatGPT	Ideación, documentación, UX writing, testing, refactor conceptual.	Acelera procesos, mejora calidad, reduce tiempos de exploración.
Gemini	Análisis de documentos, interpretación de diagramas, generación de contenido multimodal.	Excelente para trabajo con múltiples formatos y research.
Claude	Síntesis de entrevistas, PRDs, historias de usuario, análisis de feedback.	Ideal para UX Research y Product Management.
GitHub Copilot (versión cloud)	Autocompletar código, sugerencias inteligentes dentro de GitHub Codespaces.	No requiere instalación; acelera desarrollo.
Hugging Face (web)	Uso de modelos preentrenados para NLP, visión o recomendadores.	Permite prototipar IA sin infraestructura.

Herramientas de ***Inteligencia Artificial***

Herramienta	Uso en productos digitales	Impacto / bondades
Notion AI	Organización, resumen de reuniones, priorización de backlogs.	Ayuda a PMs y UX a mantener claridad y foco.
Gamma	Generación de presentaciones, casos de estudio y pitch decks.	Ideal para portafolios, storytelling y presentaciones de producto.
Canva AI	Material visual, prototipos simples, assets para portafolios.	Fácil, accesible, orientado a diseño rápido.
DALL·E	Generación de imágenes, ilustraciones, assets visuales para UI.	Facilita exploraciones visuales rápidas.

Optimización de Prompts

La optimización de prompts es un proceso esencial para obtener respuestas más precisas y relevantes. Con pequeños ajustes, puedes mejorar significativamente la calidad de las respuestas generadas por la IA.

alkemy

🏆Top 10 mejores prácticas

Técnicas para hacer prompts más eficientes

1. Asignale una personalidad y/o rol a la IA.
2. Utiliza preguntas dirigidas cuando redactes el prompt.
3. Incluye siempre información básica sobre el escenario o caso de uso.
4. Agrega detalles relevantes para evitar que el modelo haga suposiciones.
5. Establece restricciones.
6. Proporcionar ejemplos guía para ayudar al modelo a entender mejor lo que necesitas.
7. Divide tareas complejas en pasos más pequeños.
8. Registra los resultados y las acciones de mejora que hayan sido eficientes y escalables.
9. Utiliza lo que te ha funcionado en instancias anteriores como referencia.
10. Entrena a la IA para que evalúe las respuestas que te ha dado.

LIVE CODING

Demostración en vivo

Optimización de Prompts:

En esta demostración, trabajaremos paso a paso en la creación, evaluación y refinamiento de prompts utilizando técnicas prácticas para hacerlos más eficientes. Vamos a buscar:

- Diseñar prompts claros y enfocados.
- Aplicar técnicas de optimización en diferentes contextos.
- Reconocer la importancia del contexto, los detalles y la especificidad para mejorar las respuestas del modelo.

LIVE CODING

Demostración en vivo

Pasos a seguir:

1. **Crear un Prompt Básico:**

   → Prompt inicial:

   "Necesito un pseudocódigo para el proceso de entrega de un pedido en una app de delivery."

2. **Evaluar la Respuesta Inicial:**

   → Verifica si la respuesta generada por el modelo es útil, clara y funcional.

3. **Identificar Limitaciones del Prompt:**

   → ¿Qué información falta?
   → ¿Menciona todas las etapas del proceso (pedido, cocina, envío, entrega)?
   → ¿Es claro el orden de las acciones?
   → ¿El pseudocódigo tiene un inicio y un fin?

LIVE CODING

Demostración en vivo

4. **Refinar el Prompt (Iteración 1):**
   → Añade contexto y detalles:

   "Puedes escribirme un pseudocódigo simple que muestre paso a paso el proceso de un pedido en una app de comida, desde que el usuario confirma el pedido hasta que lo recibe?"

5. **Probar el Prompt Refinado:**

   → Analiza si las mejoras generan una respuesta más específica.

6. **Ir un paso más allá (Iteración 2):**

   → Prompt final (con formato):

   "Genera un pseudocódigo bien estructurado, con inicio y fin, para una app de delivery. Incluye mensajes como 'Pedido confirmado', 'Preparando comida', 'En camino' y 'Pedido entregado'"

LIVE CODING

Demostración en vivo

7. **Opcional:** Solicitar también un diagrama de flujo a partir del pseudocódigo:

   → 👉 *"¿Puedes armar un diagrama de flujo para ese proceso?"*

   → **Nota**: *esto es Ideal mostrar que la IA puede ayudarte a visualizar procesos, incluso sin saber dibujar diagramas complejos.*

8. **Comparar resultados:**

   → Evalúa cómo las iteraciones han mejorado la claridad, especificidad y funcionalidad de las respuestas.

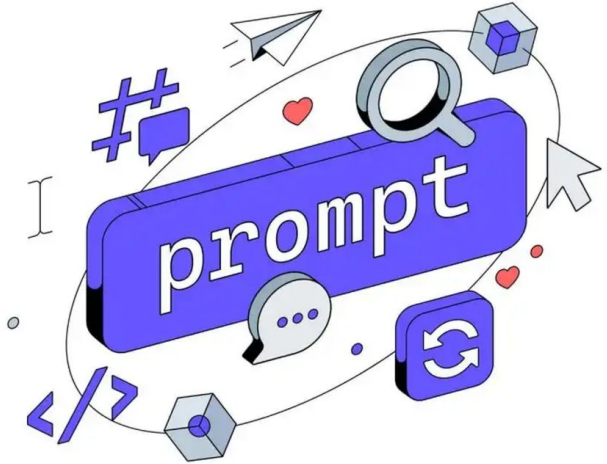
9. **Documentar el Proceso:**

   → Registra cada versión del prompt y sus respuestas, destacando las mejoras obtenidas.

IA generativa en ciberseguridad

La forma en que estructuramos los prompts **afecta directamente la calidad y relevancia de las respuestas obtenidas**. Para maximizar su efectividad, los prompts deben ser claros, concisos y específicos, pero también deben alinearse con las mejores prácticas en **privacidad y seguridad empresarial.**

Cuando utilizamos herramientas de IA en un entorno empresarial, es fundamental **proteger la privacidad de los datos** y cumplir con las políticas de seguridad.



< > Uso de prompts efectivos

Resumen

🧪 Corrección de errores

- “Este código en Python lanza un IndexError. ¿Cómo lo corrijo?”
- “Sugiere una versión más clara del mismo código (sin cambiar qué hace).”

🔧 Optimización

- “¿Cómo puedo hacer que este algoritmo tenga menor complejidad?”
- “¿Hay una versión más eficiente de este código para recorrer una lista?”

🧱 Generación

- “Genera una clase [nombre de la clase] que represente una factura con métodos para calcular impuestos.”
- “Dame un algoritmo en pseudocódigo para ordenar una lista usando el método de burbuja.”

📚 Documentación

- Crea una descripción corta para el README de [proyecto/módulo].

🧩 Traducción

- “Convierte este código de [Lenguaje 1] a [Lenguaje 2] manteniendo el mismo resultado.”

< > Comparativa de herramientas de IA populares

Herramienta	Descripción	Nivel de integración	Lenguajes soportados
ChatGPT	IA conversacional generalista que entiende código	Web/app/API	Multilenguaje
GitHub Copilot	Asistente de codificación de GitHub	Integrado en IDEs (VSCode, JetBrains)	Principalmente Python, JavaScript, Java
Tabnine	Autocompletado potenciado por IA entrenado en tu código	IDEs	Multilenguaje

< > Consideraciones éticas

Citación y respeto por la autoría

Si una herramienta de IA genera código basado en ejemplos populares o repositorios públicos, se debe procurar citar adecuadamente las fuentes si el código será reutilizado, especialmente en contextos académicos, proyectos de código abierto o comerciales. Esto refuerza una cultura de respeto hacia el trabajo de otros desarrolladores.

< > Consideraciones éticas

Evitar automatizaciones maliciosas o usos fraudulentos

La IA puede facilitar tareas como el scraping masivo de datos, el envío automatizado de solicitudes, el desarrollo de bots que infringen normas o incluso la ingeniería inversa. Estos usos deben evitarse totalmente, tanto por razones legales como éticas. Un uso responsable implica apegarse a las políticas de uso aceptable y priorizar la construcción de soluciones seguras y legítimas.

< > Consideraciones éticas

Reconocer las limitaciones de la IA

La IA no es infalible. Puede generar código ineficiente, vulnerable o simplemente incorrecto. Incluso cuando el resultado parece coherente o funcional, no debe considerarse una verdad absoluta. Es fundamental que el desarrollador mantenga una actitud crítica y use herramientas de testing, análisis estático y revisión de código como respaldo.

< > Beneficios concretos

Incremento de la productividad del desarrollador

El uso inteligente de herramientas de IA permite acelerar significativamente el desarrollo de software. Desde la generación de funciones repetitivas, la documentación de APIs, hasta la automatización de pruebas unitarias, la IA libera tiempo que puede emplearse en tareas de mayor valor.

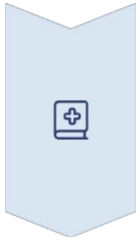
Mejora en la calidad del código

Muchos modelos de IA están entrenados con buenas prácticas de programación, lo que puede ayudar a los desarrolladores a adoptar patrones correctos de diseño, nombres de variables descriptivos y estructuras lógicas limpias. Esto contribuye a escribir código más legible, mantenible y menos propenso a errores.

Aceleración del aprendizaje técnico

Las herramientas de IA pueden actuar como tutores interactivos. Explican errores, sugieren soluciones alternativas y permiten a los principiantes comprender conceptos complejos en un lenguaje natural. Así, los desarrolladores en formación pueden aprender mientras construyen y depuran código real.

< > Beneficios concretos

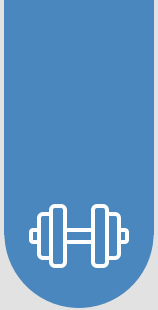


Documentación automatizada

La generación de documentación técnica es una tarea esencial pero a menudo relegada. Las IA pueden generar descripciones automáticas de funciones, estructuras de datos o flujos de trabajo, lo que garantiza mayor coherencia en los proyectos y facilita la incorporación de nuevos miembros al equipo.

Apoyo en debugging y testing

Al identificar errores en el código, muchos asistentes de IA pueden sugerir posibles causas o soluciones. También pueden ayudar a construir pruebas unitarias o de integración a partir del código existente, lo que mejora la cobertura de pruebas y reduce el tiempo dedicado a tareas manuales.



Ejercicio N° 1

¿Es un número primo?

¿Es un número primo?

Contexto: 🙌

Aplicarás lo aprendido para desarrollar un algoritmo completo que determine si un número es primo, usando tanto pseudocódigo como herramientas de simulación.

Consigna: ✍️

1. Diseña un algoritmo en pseudocódigo que reciba un número y determine si es primo.
2. Representa su lógica con un diagrama de flujo.
3. Ejecuta paso a paso en PSeInt para observar el flujo y estado de variables.
4. Analiza: ¿Qué estructuras de control usaste? ¿Tu algoritmo es eficiente?

Tiempo 🕐**:** *20 Minutos*

Consideraciones: ⚙️

- Usa estructuras de repetición como Para o Mientras.
- Considera los criterios de finitud y determinismo del algoritmo.
- Verifica cómo cambian las variables en cada paso.

¿Alguna consulta?

alkemy

Resumen

¿Qué logramos en esta clase?

- ✓ ***Entendimos qué es un algoritmo y su valor en la resolución de problemas.***
- ✓ ***Aprendimos a estructurar pseudocódigo y a usar palabras clave correctamente.***
- ✓ ***Utilizamos PSeInt para ver la ejecución paso a paso y visualizar errores lógicos.***
- ✓ ***Creamos diagramas de flujo para representar visualmente los pasos del algoritmo.***
- ✓ ***Repasamos las principales herramientas de IA gratuitas***
- ✓ ***Entendimos las diferentes formas de escribir prompts.***

¡Ponte a prueba!

Momento de ejercitación

Te invitamos a aprovechar esta última sección del espacio sincrónico para realizar de manera individual las ***actividades disponibles en la plataforma.*** *Estas propuestas son clave para afianzar lo trabajado y* ***forman parte obligatoria del recorrido de aprendizaje.***

💪 Análisis de caso ——— 💪 Selección Múltiple

💪 Comprensión lectora

Si al resolverlas surge alguna duda, compártela o tráela al próximo encuentro sincrónico.

#Checkout

¿Qué les pareció la clase de hoy?

3. Estuvo bien, pero podría mejorar
¿Qué se puede potenciar?

4. ¡Estuvo bien!
¿Qué me llevo?

2. No me convenció mucho
¿Qué no te gustó de la clase?

5. ¡Me encantó!
¿Qué fue lo mejor de la clase?

1. Red flag
¿Qué te hubiera gustado que fuera diferente?

¡Muchas gracias!

alkemy