Rise & Code

Un libro de programación para todos

Open Source Community

March 17, 2025

Contents

Rise & Code

Un libro de programación para todos

Acerca de este libro

"Rise & Code" hace que el emocionante mundo de la programación sea accesible para todos, independientemente de su edad, origen o acceso a la tecnología. A través de lecciones interactivas, imágenes atractivas y una metodología única de cuaderno, ofrece un enfoque fresco y empoderador para aprender a programar.

Características principales

- **No se requiere computadora**: Aprende conceptos de programación usando solo papel y lápiz
- **Actividades prácticas**: Cada capítulo incluye ejercicios y actividades prácticas

- **Aprendizaje visual**: Conceptos ilustrados a través de diagramas y diagramas de flujo
- **Plan de estudios progresivo**: Desarrolla habilidades gradualmente desde temas fundamentales hasta avanzados
- **Diseño inclusivo**: Creado para audiencias diversas con diferentes estilos de aprendizaje

Nuestra misión

- Hacer que la educación en programación sea accesible para comunidades desatendidas
- Enseñar pensamiento computacional a través de actividades sin conexión
- Construir habilidades fundamentales que se transfieran a cualquier lenguaje de programación
- Crear un recurso que pueda ser compartido, impreso y distribuido libremente

Licencia

Levántate y Codifica

Un Libro de Programación para Todos

Version: v2025.03.17-0126

Generated on: March 17, 2025

Este libro está diseñado para enseñar programación, desarrollo de software y resolución de problemas lógicos a personas sin acceso a computadoras.

Capítulo 1: Introducción - El mundo de la programación sin computadora

¡Bienvenido al primer capítulo de "Rise & Code"! Este capítulo te introduce al concepto de programación sin computadora y prepara el escenario para el resto del libro.

Objetivos del capítulo

- Entender por qué las habilidades de programación son valiosas en el mundo actual
- Reconocer que los conceptos de programación pueden aprenderse sin una computadora
- Aprender a usar el método de cuaderno para practicar conceptos de programación
- Familiarizarse con el enfoque y la estructura del libro

Secciones

1. Por qué importa la programación
2. A quién está dirigido este libro
3. Cómo usar este libro (incluido el método de cuaderno)

Actividades

- Tu primer algoritmo
- Identificar el pensamiento computacional en la vida cotidiana
- Configurar tu cuaderno de programación

Resumen del capítulo

¿Listo para revisar lo que has aprendido? Consulta el Resumen del Capítulo para repasar los conceptos clave y obtener una vista previa de lo que viene a continuación.

why programming matters

Por qué importa la programación

Introducción

Imagina un mundo donde pudieras decirle a una máquina exactamente lo que quieres que haga, y lo hiciera perfectamente cada vez. Ese es el poder de la programación. Pero la programación es mucho más que simplemente controlar computadoras—se trata de desarrollar una forma de pensar que te ayuda a resolver todo tipo de problemas.

¿Qué es la programación?

En su esencia, la programación consiste en dar instrucciones a una computadora. Pero a diferencia de los humanos, las computadoras necesitan instrucciones extremadamente precisas. Siguen exactamente lo que les dices que hagan—ni más, ni menos.

Piensa en dar direcciones a un amigo versus a un extraño en una nueva ciudad. Con tu amigo, podrías decir: "Nos vemos en el café de siempre". El amigo completa los vacíos usando conocimiento compartido. Pero con un extraño, necesitas proporcionar cada detalle: qué calles tomar, puntos de referencia a observar y exactamente cómo reconocer el café.

La programación es como dar direcciones a ese extraño—que además resulta ser increíblemente rápido, nunca se cansa y sigue tus instrucciones exactamente.

La programación en la vida cotidiana

Es posible que no te des cuenta, pero estás rodeado de programación todos los días:

- Cuando retiras dinero de un cajero automático
- Cuando los semáforos cambian según la hora del día
- Cuando tu teléfono móvil te notifica sobre un mensaje
- Cuando los agricultores usan sistemas de riego automatizados
- Cuando los trabajadores de salud rastrean brotes de enfermedades

Todos estos sistemas funcionan con instrucciones escritas por programadores. Y cada vez más, saber programar—o al menos entender cómo funciona la programación—se está convirtiendo en una habilidad esencial para muchos trabajos y oportunidades.

¿Por qué aprender programación sin una computadora?

Tal vez te estés preguntando: "¿Cómo puedo aprender programación sin una computadora?" ¡Es una pregunta válida!

who this book is for

A quién está dirigido este libro

Introducción

La programación es para todos. Ya seas estudiante, maestro, agricultor, médico, comerciante o simplemente alguien con curiosidad sobre cómo funciona la tecnología, este libro está diseñado para ti. No necesitas experiencia previa con computadoras o programación para beneficiarte de los conceptos que exploraremos juntos.

Estudiantes de todas las edades

¿Eres una persona joven que quiere entender la tecnología que da forma a nuestro mundo? ¿O tal vez eres un adulto que nunca tuvo la oportunidad de aprender programación antes? Este libro es para estudiantes de todas las edades.

Las actividades y explicaciones están diseñadas para ser accesibles tanto si tienes 10 como 70 años. Algunos conceptos pueden ser desafiantes al principio, pero los hemos dividido en piezas manejables que se construyen una sobre otra.

Personas con acceso limitado a la tecnología

Uno de los objetivos más importantes de este libro es hacer que la educación en programación sea accesible para todos, especialmente para aquellos que no tienen acceso regular a computadoras.

Si tú: - Vives en una región con electricidad o acceso a internet limitados - No tienes una computadora personal - Compartes recursos tecnológicos limitados con muchos otros - Solo tienes acceso ocasional a computadoras

...entonces este libro fue creado específicamente pensando en ti. Todo en este libro se puede hacer solo con papel, lápiz y tu propio pensamiento creativo.

Educadores y líderes comunitarios

Si eres maestro, mentor o líder comunitario que busca introducir conceptos de programación a tus estudiantes o miembros de la comunidad, este libro proporciona un marco para hacerlo sin requerir un laboratorio de computación o equipo costoso.

Las actividades pueden adaptarse para: - Entornos de aula - Programas extraescolares - Talleres comunitarios - Mentoría individual

Cada capítulo incluye actividades que se pueden realizar individualmente o en grupos, haciéndolas flexibles para diferentes entornos de aprendizaje.

Aprendices autodidactas

Si estás aprendiendo por tu cuenta, este libro proporciona un camino estructurado para aprender conceptos de programación a tu propio ritmo. Las actividades están diseñadas para ser autónomas, con instrucciones claras y preguntas reflexivas para guiar tu aprendizaje.

Aquellos que aprenden haciendo

Muchas personas aprenden mejor a través de actividades prácticas y manuales que mediante teoría abstracta. Si eres alguien que prefiere aprender haciendo, encontrarás que este libro está lleno de ejercicios interactivos que te permiten practicar conceptos de programación inmediatamente.

Cualquiera con curiosidad sobre cómo funciona la programación

Quizás has oído hablar de programación y quieres entender de qué se trata. O tal vez usas tecnología todos los días y quieres echar un vistazo detrás del telón para ver cómo funciona. Este libro te ayudará a desmitificar la programación y te mostrará el pensamiento lógico que impulsa nuestro mundo digital.

Lo que NO necesitas para este libro

Es importante destacar lo que NO necesitas para este libro:

- No necesitas una computadora
- No necesitas acceso a internet
- No necesitas materiales costosos
- No necesitas experiencia previa con matemáticas o programación
- No necesitas hablar ningún idioma específico (aunque este libro está actualmente disponible en [lista de idiomas aquí])

Lo que SÍ necesitas para este libro

Esto es todo lo que necesitas para comenzar:

- Un cuaderno o papel
- Algo para escribir (lápiz, bolígrafo, etc.)
- Curiosidad y disposición para probar nuevas formas de pensar
- Persistencia cuando surjan desafíos (¡y surgirán!)
- Tiempo para trabajar en actividades y reflexionar sobre lo que has aprendido

Actividad: Tu perfil de aprendizaje

Materiales necesarios: Tu cuaderno y algo para escribir

Instrucciones: 1. Titula una página en tu cuaderno “Mi perfil de aprendizaje” 2. Responde las siguientes preguntas: - ¿Qué te interesa más sobre la programación? - ¿Qué experiencia (si alguna) tienes con la tecnología? - ¿Cómo aprendes mejor? (Leyendo, haciendo, viendo, discutiendo, etc.) - ¿Qué tipo de problemas te gustaría resolver con la programación? - ¿Qué podría ser desafiante para ti al aprender a programar? - ¿Qué recursos y apoyo tienes disponibles? 3. Mantén esta página a mano mientras trabajas con el libro. Te ayudará a conectar los conceptos con tus propios objetivos y estilo de aprendizaje.

Puntos clave

- Este libro está diseñado para personas de todas las edades y orígenes
- No se requiere acceso a computadoras para aprender de este libro
- Todos pueden beneficiarse de los conceptos de programación, independientemente de su carrera o intereses
- Diferentes lectores tendrán diferentes objetivos y preferencias de aprendizaje
- La programación es una habilidad que cualquiera puede aprender con práctica y persistencia

En la siguiente sección, explicaremos cómo usar este libro de manera efectiva, incluyendo el método de cuaderno que será central para tu experiencia de aprendizaje.

Cómo usar este libro (incluido el método de cuaderno)

Introducción

Este libro está diseñado para ser tanto una guía como un cuaderno de trabajo para tu viaje de programación. En esta sección, explicaremos cómo aprovechar al máximo el libro e introduciremos el "método de cuaderno", un enfoque poderoso para aprender programación sin una computadora.

El método de cuaderno: Tu computadora de papel

A lo largo de la historia, grandes científicos, matemáticos e inventores han utilizado cuadernos para desarrollar sus ideas. Leonardo da Vinci llenó miles de páginas con bocetos, cálculos y observaciones. Ada Lovelace, a menudo considerada la primera programadora del mundo, usó papel para escribir el primer algoritmo diseñado para una máquina.

Siguiendo esta tradición, tu cuaderno se convertirá en tu "computadora de papel", un lugar donde puedes trabajar con conceptos de programación, seguir tu progreso, probar tus ideas y reflexionar sobre tu aprendizaje.

Configurando tu cuaderno de programación

Para comenzar, necesitarás un cuaderno que se convertirá en tu compañero de programación. Idealmente, elige uno que:

- Tenga muchas páginas (al menos 100)
- Se mantenga abierto fácilmente cuando se coloca en una superficie plana
- Tenga páginas lo suficientemente grandes para diagramas
- Tenga papel en blanco o papel cuadriculado (en lugar de líneas) si es posible, pero el papel con líneas también funciona

Si no tienes un cuaderno dedicado, puedes usar hojas sueltas recopiladas en una carpeta, o incluso hacer tu propio cuaderno uniendo hojas de papel.

Organizando tu cuaderno

Recomendamos dividir tu cuaderno en estas secciones:

1. **Tabla de contenidos** (primeras páginas)
    - Deja espacio para registrar lo que has escrito y dónde encontrarlo
2. **Conceptos** (aproximadamente 25% de tu cuaderno)
    - Para notas sobre conceptos de programación a medida que los aprendes
    - Incluye tus propias explicaciones y ejemplos

3. **Ejercicios y actividades** (aproximadamente 50% de tu cuaderno)
    - Para completar las actividades del libro
    - Trabajar en tus propios problemas de práctica
    - Espacio para depurar y revisar tu trabajo
4. **Reflexiones** (aproximadamente 15% de tu cuaderno)
    - Registra lo que has aprendido
    - Anota conexiones entre conceptos
    - Realiza un seguimiento de desafíos y avances
    - Establece metas para lo que aprenderás a continuación
5. **Referencia** (aproximadamente 10% de tu cuaderno)
    - Crea tus propias guías de referencia rápida
    - Mantén un registro de símbolos, términos y conceptos que quieras recordar

Puedes marcar estas secciones con pestañas, marcadores o coloreando los bordes de las páginas.

Cómo está estructurado el libro

"Rise & Code" está organizado en capítulos que se construyen uno sobre otro. Cada capítulo sigue una estructura similar:

1. **Introducción** - Visión general de los conceptos cubiertos
2. **Conceptos básicos** - Explicaciones de ideas clave de programación
3. **Ejemplos** - Ilustraciones de conceptos en acción
4. **Actividades** - Ejercicios prácticos para practicar los conceptos
5. **Reflexiones** - Preguntas para profundizar tu comprensión
6. **Puntos clave** - Resumen de los puntos principales
7. **Próximos pasos** - Vista previa de lo que viene a continuación

Aunque el libro está diseñado para leerse en orden, siéntete libre de saltar a temas específicos si ya te sientes cómodo con los conceptos anteriores.

Trabajando con las actividades

Las actividades son el corazón de este libro. Están diseñadas para realizarse con materiales simples y para involucrarte activamente en el proceso de aprendizaje. Para cada actividad:

1. **Lee completamente** antes de comenzar
2. **Reúne los materiales necesarios** (generalmente solo tu cuaderno y algo para escribir)
3. **Trabaja a tu propio ritmo** - algunas actividades pueden tomar minutos, otras pueden tomar una hora o más
4. **Registra tu trabajo** en tu cuaderno
5. **Reflexiona sobre el proceso** y lo que has aprendido
6. **Comprueba tu comprensión** usando las preguntas proporcionadas
7. **Revisa las actividades desafiantes** más tarde si es necesario

No te preocupes por cometer errores, ¡son parte del proceso de aprendizaje! De hecho, encontrar y corregir errores (llamado "depuración" en programación) es una de las habilidades más importantes que desarrollarás.

Aprendiendo solo vs. aprendiendo en grupo

Este libro puede usarse eficazmente tanto por tu cuenta como en grupo:

Para estudiantes individuales: - Establece un horario regular para trabajar con el libro - Encuentra formas de probar tu comprensión explicando conceptos en voz alta - Crea tus propios desafíos para ampliar las actividades - Conéctate con otros que estén aprendiendo a programar si es posible, incluso a distancia

Para grupos: - Túrnense para explicarse conceptos entre ustedes - Trabajen en actividades de forma colaborativa - Discutan diferentes enfoques para resolver problemas - Creen grupos de estudio para compartir desafíos e ideas

Seguimiento de tu progreso

A medida que trabajas con el libro, es útil hacer un seguimiento de tu progreso:

- Marca las secciones completadas en la tabla de contenidos
- Anota los conceptos que te resulten desafiantes y que quizás quieras revisar
- Celebra tus momentos de "¡ajá!" en tu sección de reflexiones
- Revisa periódicamente el trabajo anterior para ver cuánto has avanzado
- Establece metas para lo que quieres aprender a continuación

Actividad: Configurando tu cuaderno de programación

Materiales necesarios: Un cuaderno o papel, algo para escribir y, opcionalmente, marcadores/pestañas para dividir secciones

Instrucciones: 1. Configura las cinco secciones en tu cuaderno como se describió anteriormente. 2. En la primera página, escribe tu nombre y "Mi viaje de programación" o tu propio título. 3. Crea una tabla de contenido simple con espacio para agregar entradas a medida que avanzas. 4. En la primera página de tu sección de Conceptos, escribe la fecha de hoy y "Comenzando mi viaje de programación". 5. Escribe de 3 a 5 cosas que esperas aprender de este libro. 6. En la sección de Referencia, crea tu primera entrada: un dibujo de un diagrama de flujo simple que muestre los pasos para tomar una decisión (como qué comer para la cena).

Consejos para el éxito

1. **La consistencia importa más que la duración** - Incluso 15 minutos de práctica diaria es mejor que varias horas una vez por semana.
2. **Explica los conceptos a otros** (o a ti mismo) - Enseñar algo es una de las mejores formas de aprenderlo.

3. **Dibuja imágenes** - Las representaciones visuales ayudan a solidificar conceptos abstractos.
4. **Conecta ideas con tus propias experiencias** - ¿Cómo se relaciona un concepto de programación con algo que ya conoces?
5. **Ten paciencia contigo mismo** - La programación implica una nueva forma de pensar que toma tiempo desarrollar.
6. **Revisa regularmente** - Vuelve a conceptos anteriores para profundizar tu comprensión.
7. **Aplica conceptos ampliamente** - Trata de ver cómo las ideas de programación se relacionan con situaciones cotidianas.

Puntos clave

- Tu cuaderno es tu herramienta más importante para aprender programación sin una computadora
- Organizar tu proceso de aprendizaje te ayuda a retener y construir sobre conceptos
- Las actividades proporcionan práctica práctica esencial para aprender programación
- La práctica consistente, la reflexión y la aplicación son clave para el éxito
- El libro está diseñado para ser flexible para diferentes estilos y contextos de aprendizaje

En el próximo capítulo, nos sumergiremos en los fundamentos de la lógica y la estructura, los bloques de construcción de todos los lenguajes de programación.

Activities

first algorithm

Actividad: Tu primer algoritmo

Visión general

Esta actividad te introduce a la creación de algoritmos—instrucciones paso a paso para resolver un problema. Practicarás descomponer una tarea familiar en pasos claros y precisos que podrían ser seguidos por alguien que nunca ha realizado la tarea antes.

Objetivos de aprendizaje

- Entender qué es un algoritmo
- Practicar la escritura de instrucciones claras y precisas
- Aprender a descomponer tareas complejas en pasos simples
- Identificar suposiciones en las instrucciones

Materiales necesarios

- Cuaderno y lápiz
- Opcional: lápices o marcadores de colores

Tiempo requerido

30-45 minutos

Instrucciones

Parte 1: Elige tu tarea

1. Selecciona una tarea simple y cotidiana que sabes cómo hacer. Algunas ideas:
   - Hacer un sándwich
   - Cepillarse los dientes
   - Atar cordones de zapatos
   - Dibujar una forma simple
   - Plantar una semilla
2. En tu cuaderno, escribe el nombre de tu tarea en la parte superior de la página.

Parte 2: Escribe tu algoritmo

1. Piensa en cómo completar tu tarea, desglosándola en pasos individuales.
2. Escribe cada paso en orden, comenzando con el paso 1.
3. Sé lo más claro y específico posible. Imagina que estás escribiendo instrucciones para alguien que nunca ha realizado esta tarea antes.

4. Intenta incluir al menos 10 pasos.

Parte 3: Prueba tu algoritmo

1. Revisa tu algoritmo y busca pasos faltantes o suposiciones.
2. Si es posible, pide a un amigo o familiar que siga tus instrucciones exactamente como están escritas. Obsérvalos sin proporcionar ninguna guía adicional.
3. Anota cualquier punto donde se confundan o donde tus instrucciones no fueran lo suficientemente claras.

Parte 4: Depura y mejora

1. Basándote en tus observaciones, revisa tu algoritmo para solucionar cualquier problema.
2. Agrega pasos donde sea necesario y aclara instrucciones ambiguas.
3. Escribe tu algoritmo mejorado en una nueva página de tu cuaderno.

Parte 5: Reflexiona

Responde estas preguntas en tu cuaderno: 1. ¿Cuál fue la parte más desafiante de escribir tu algoritmo? 2. ¿Hiciste alguna suposición en tus instrucciones originales? ¿Cuáles fueron? 3. ¿En qué se parece y en qué se diferencia escribir un algoritmo de dar indicaciones a una persona? 4. ¿Por qué crees que las computadoras necesitan instrucciones más precisas que los humanos?

Ejemplo

Aquí hay un ejemplo de un algoritmo para hacer una taza de té:

1. Consigue una taza limpia
2. Consigue una bolsa de té
3. Llena una tetera con agua
4. Coloca la tetera en la estufa
5. Enciende la estufa a fuego alto
6. Espera hasta que el agua hierva
7. Apaga la estufa
8. Vierte el agua caliente en la taza, llenándola aproximadamente a 2 cm del borde
9. Coloca la bolsa de té en la taza
10. Espera 3 minutos para que el té se infusione
11. Retira la bolsa de té de la taza
12. (Opcional) Agrega azúcar o leche según tu gusto
13. Revuelve el té

Actividades de extensión

- Intenta reescribir tu algoritmo con el mínimo número posible de pasos mientras mantienes la claridad
- Escribe un algoritmo para una audiencia diferente (por ejemplo, un niño, un adulto, un robot)
- Dibuja símbolos o diagramas para representar cada paso en tu algoritmo
- Encuentra una receta publicada y analízala como un algoritmo. ¿Podrías mejorarla?

Conexión con la programación

En programación, los algoritmos son el corazón de cada programa. Los programadores necesitan descomponer problemas en pasos claros y lógicos que una computadora pueda seguir. Las habilidades que practicaste en esta actividad—claridad, precisión y atención al detalle—son exactamente lo que necesitas para escribir programas de computadora efectivos.

computational thinking in everyday life

Actividad: Pensamiento computacional en la vida cotidiana

Visión general

Esta actividad te ayudará a reconocer cómo el pensamiento computacional está presente en tu vida diaria, incluso cuando no estás usando una computadora. Identificarás ejemplos de los cuatro componentes principales del pensamiento computacional: descomposición, reconocimiento de patrones, abstracción y algoritmos.

Objetivos de aprendizaje

- Identificar ejemplos de pensamiento computacional en actividades cotidianas
- Comprender los cuatro componentes del pensamiento computacional
- Reconocer cómo estas habilidades de pensamiento son útiles más allá de la programación

Materiales necesarios

- Cuaderno y lápiz
- Opcional: lápices o marcadores de colores para organizar notas

Tiempo requerido

45-60 minutos

Instrucciones

Parte 1: Introducción al pensamiento computacional

Primero, revisa los cuatro componentes del pensamiento computacional:

1. **Descomposición**: Dividir un problema complejo en partes más pequeñas y manejables.
2. **Reconocimiento de patrones**: Identificar similitudes o tendencias dentro o entre problemas.
3. **Abstracción**: Enfocarse en la información importante, ignorando detalles irrelevantes.
4. **Algoritmos**: Desarrollar soluciones paso a paso o reglas para resolver problemas.

En tu cuaderno, crea cuatro secciones, una para cada componente, donde registrarás tus observaciones.

Parte 2: Observación

Durante un día completo, observa tus actividades y las de quienes te rodean. Busca ejemplos de los cuatro componentes del pensamiento computacional en acción.

Por ejemplo: - Al cocinar, ¿cómo divides la preparación en tareas más pequeñas? (descomposición) - ¿Qué patrones has notado en tu rutina matutina? (reconocimiento de patrones) - Cuando explicas una ubicación a alguien, ¿qué detalles incluyes y cuáles omites? (abstracción) - ¿Qué procesos sigues paso a paso, como una receta o un procedimiento de limpieza? (algoritmos)

Registra al menos dos ejemplos para cada componente.

Parte 3: Análisis

Una vez que hayas recopilado tus ejemplos, para cada uno:

1. Describe la situación o actividad
2. Explica qué componente de pensamiento computacional observaste
3. Detalla cómo este componente ayudó a resolver un problema o realizar una tarea
4. Reflexiona sobre cómo esta misma habilidad podría aplicarse a la programación

Parte 4: Conexiones más profundas

Elige una actividad que realizas regularmente (cocinar, hacer ejercicio, ir al trabajo, organizar tu espacio, etc.) y analízala usando los cuatro componentes del pensamiento computacional:

1. ¿Cómo la descompones en subtareas?
2. ¿Qué patrones has notado o creado?
3. ¿Qué abstracciones usas? (atajos mentales, reglas generales, etc.)
4. ¿Qué algoritmos o procedimientos sigues?

Escribe tu análisis en tu cuaderno, detallando cómo estos cuatro componentes trabajando juntos te ayudan a completar la actividad de manera efectiva.

Parte 5: Reflexión

Responde estas preguntas en tu cuaderno:

1. ¿Qué componente del pensamiento computacional te resulta más natural usar? ¿Por qué?
2. ¿Qué componente te parece más desafiante? ¿Por qué?
3. ¿Cómo crees que desarrollar estas habilidades de pensamiento podría beneficiarte en:
   - Tus estudios o trabajo?
   - Resolución de problemas cotidianos?

- Aprender a programar?

4. ¿Observaste cómo diferentes personas podrían usar diferentes enfoques de pensamiento computacional para la misma tarea?

Ejemplo

Situación: Organizar una reunión familiar grande

Descomposición: - Dividir en tareas: crear lista de invitados, planificar comida, decoración, actividades, etc. - Cada tarea se divide aún más: la lista de comida se separa en aperitivos, plato principal, postres, bebidas

Reconocimiento de patrones: - Notar que ciertos familiares siempre llegan tarde y planificar en consecuencia - Reconocer qué combinaciones de alimentos han funcionado bien en eventos anteriores

Abstracción: - Crear categorías generales de necesidades dietéticas en lugar de rastrear cada preferencia individual - Desarrollar un tema general que guíe las decisiones en lugar de considerar cada elemento decorativo individualmente

Algoritmos: - Seguir un cronograma para la preparación del evento: 2 semanas antes, 1 semana antes, día anterior, día del evento - Procedimiento para configurar mesas y asientos de manera eficiente

Actividades de extensión

- Entrevista a amigos o familiares sobre cómo usan el pensamiento computacional en sus trabajos o pasatiempos
- Elige un proceso en tu escuela, trabajo o comunidad que podría mejorarse aplicando el pensamiento computacional
- Crea un póster o diagrama visual que ilustre cómo los cuatro componentes del pensamiento computacional trabajan juntos

Conexión con la programación

Los programadores constantemente utilizan estos cuatro componentes del pensamiento computacional para crear software. A medida que aprendas a programar: - Descompondrás grandes proyectos en funciones y módulos más pequeños - Reconocerás patrones que te permitirán reutilizar y adaptar código - Crearás abstracciones como variables y funciones para ocultar complejidad - Desarrollarás algoritmos que resuelvan problemas específicos de manera eficiente

Desarrollar estas habilidades de pensamiento ahora te dará una base sólida para la programación, incluso antes de escribir tu primera línea de código.

setting up your coding notebook

Actividad: Configurando tu cuaderno de programación

Visión general

Esta actividad te guiará a través del proceso de configuración de tu cuaderno de programación, que será tu herramienta principal para aprender a lo largo de este libro. A diferencia de otros cursos de programación que requieren una computadora, usarás este cuaderno como tu "computadora de papel" para trabajar con conceptos de programación, resolver problemas y registrar tu progreso.

Objetivos de aprendizaje

- Crear un sistema organizado para el aprendizaje de programación
- Establecer prácticas de documentación que apoyen el pensamiento computacional
- Desarrollar un espacio personal para experimentar con conceptos de programación

Materiales necesarios

- Un cuaderno nuevo (preferiblemente con al menos 100 páginas)
- Bolígrafos, lápices, marcadores o lápices de colores
- Regla (opcional)
- Notas adhesivas o marcadores de página (opcional)

Tiempo requerido

30-45 minutos

Instrucciones

Parte 1: Eligiendo tu cuaderno

Si aún no lo has hecho, consigue un cuaderno que usarás exclusivamente para tu viaje de programación.

Consideraciones para elegir un buen cuaderno: - **Tamaño**: Lo suficientemente grande para escribir cómodamente código y diagramas (A4 o carta es ideal) - **Encuadernación**: Una que permita que el cuaderno se abra completamente y quede plano - **Páginas**: Preferiblemente cuadriculadas o con líneas para facilitar la escritura de código y dibujar diagramas - **Durabilidad**: Una cubierta resistente que proteja tu trabajo

Parte 2: Creando una tabla de contenidos

1. Reserva las primeras 2-4 páginas de tu cuaderno para una tabla de contenidos.
2. Crea una tabla con tres columnas:
    - Número de página
    - Título/Tema
    - Fecha
3. Decora la primera página con un título como "Mi viaje de programación" o "Cuaderno de código" y personalízalo como desees.
4. A medida que avances en el libro, actualiza esta tabla de contenidos para facilitar la referencia futura.

Parte 3: Dividiendo tu cuaderno en secciones

Divide tu cuaderno en las siguientes secciones. Usa notas adhesivas, marcadores de página o simplemente marca las páginas con títulos claros:

1. **Conceptos y notas** (aproximadamente 40% de tu cuaderno)
    - Aquí registrarás definiciones, explicaciones y notas sobre conceptos de programación
    - Incluye ejemplos que te ayuden a entender cada concepto
    - Usa colores o símbolos para resaltar puntos importantes
2. **Ejercicios y soluciones** (aproximadamente 30% de tu cuaderno)
    - Dedica esta sección a resolver los ejercicios y desafíos del libro
    - Incluye tanto tus intentos de solución como tus soluciones finales
    - Documenta los errores que encuentres y cómo los resolviste
3. **Diagramas y visualizaciones** (aproximadamente 15% de tu cuaderno)
    - Usa esta sección para crear diagramas de flujo, mapas mentales y otras representaciones visuales
    - Dibuja cómo los diferentes conceptos se relacionan entre sí
    - Visualiza algoritmos y estructuras de datos
4. **Reflexión y aprendizaje** (aproximadamente 15% de tu cuaderno)
    - Registra tus pensamientos sobre lo que estás aprendiendo
    - Anota preguntas que surjan durante tu estudio
    - Documenta tus "momentos ajá" y descubrimientos
    - Haz seguimiento de tu progreso y establece metas futuras

Parte 4: Configurando tu primera página de contenido

1. Después de la tabla de contenidos, crea tu primera página de contenido.
2. Escribe la fecha y un título: "Capítulo 1: Introducción a la programación".
3. Debajo, escribe un breve párrafo sobre por qué estás interesado en aprender a programar y qué esperas lograr.
4. Deja espacio para agregar tus principales aprendizajes de este capítulo.

Parte 5: Estableciendo convenciones de codificación

1. Crea una página llamada "Mis convenciones de codificación" donde documentarás cómo representarás diferentes elementos de programación en tu cuaderno.
2. Decide cómo distinguirás visualmente:
   - Nombres de variables
   - Funciones y métodos
   - Palabras clave del lenguaje
   - Comentarios y explicaciones
   - Resultados de código

Por ejemplo: - Variables: Subrayadas - Funciones: En negritas o con un color específico - Comentarios: En cursiva o entre paréntesis - Resultados: En un recuadro o con un color diferente

Parte 6: Reflexión

En la sección de reflexión de tu cuaderno, responde las siguientes preguntas:

1. ¿Qué te entusiasma más sobre aprender a programar usando este método de cuaderno?
2. ¿Qué desafíos anticipas al aprender programación sin una computadora?
3. ¿Cómo planeas mantener tu cuaderno organizado a medida que acumulas más conocimiento?
4. ¿Qué hábitos de estudio crees que serán importantes para tener éxito con este enfoque?

Consejos para el éxito

- **Sé consistente**: Dedica tiempo regularmente a tu cuaderno, idealmente a la misma hora cada día o semana.
- **Sé visual**: Usa colores, diagramas y símbolos para hacer que tu cuaderno sea visualmente atractivo y más fácil de revisar.
- **Sé minucioso**: Escribe claramente y documenta tu pensamiento. Esto te ayudará cuando revises material más adelante.
- **Sé paciente**: La programación en papel puede parecer más lenta, pero ofrece una comprensión más profunda de los conceptos fundamentales.
- **Sé creativo**: Adapta este sistema a tu estilo de aprendizaje y modifícalo según sea necesario.

Extensiones

- Crea una "guía de referencia rápida" en la contratapa de tu cuaderno para conceptos que usas frecuentemente
- Diseña un sistema de símbolos o iconos para marcar diferentes tipos de entradas (problemas por resolver, conceptos confusos, ideas para explorar, etc.)

- Reserva algunas páginas al final de tu cuaderno para un glosario de términos de programación
- Incluye una sección "ideas de proyectos" para anotar ideas de programación que te gustaría implementar en el futuro

Conexión con la programación

Los programadores profesionales suelen mantener registros detallados de su trabajo, incluyendo registros de decisiones, documentación y notas sobre problemas y soluciones. Tu cuaderno de programación te ayuda a desarrollar estas habilidades de documentación que son esenciales en el campo de la tecnología, mientras te permite explorar conceptos de programación de una manera táctil y reflexiva.

Chapter Summary

Capítulo 1: Resumen

Puntos clave

Por qué la programación importa

- La programación es una habilidad fundamental en el mundo digital actual
- Aprender a programar desarrolla el pensamiento crítico y las habilidades para resolver problemas
- El pensamiento computacional puede aplicarse a muchas áreas fuera de la programación
- Incluso la programación básica proporciona acceso a tecnologías y oportunidades nuevas

Para quién es este libro

- Este libro está diseñado para ser accesible a todos, independientemente de su acceso a tecnología
- No se requiere una computadora, solo un cuaderno y un lápiz
- El contenido se adapta a estudiantes de todas las edades y orígenes
- Los conceptos se centran en fundamentos universales, no en lenguajes o tecnologías específicas

Cómo usar este libro

- El método del cuaderno proporciona una forma táctil y deliberada de aprender programación
- La estructura del libro incluye conceptos, actividades y reflexiones para un aprendizaje completo
- Se pueden elegir diferentes caminos a través del material en función de los intereses y necesidades
- La consistencia y la práctica regular son clave para el éxito

Conceptos introducidos

- **Algoritmos**: Instrucciones paso a paso para realizar una tarea
- **Pensamiento computacional**: Un enfoque para resolver problemas que incluye:
  - Descomposición: Dividir problemas grandes en partes más pequeñas
  - Reconocimiento de patrones: Identificar tendencias y similitudes
  - Abstracción: Enfocarse en lo importante, ignorando detalles irrelevantes
  - Algoritmos: Crear soluciones paso a paso
- **Programación sin computadoras**: Cómo aprender y practicar conceptos de programación usando solo papel y lápiz

Actividades completadas

1. **Tu primer algoritmo**: Creación de instrucciones paso a paso para una tarea cotidiana
2. **Pensamiento computacional en la vida cotidiana**: Identificación de ejemplos de pensamiento computacional en el mundo que te rodea
3. **Configurando tu cuaderno de programación**: Establecimiento de un sistema organizado para documentar tu aprendizaje

Próximos pasos

En el Capítulo 2, profundizaremos en los conceptos fundamentales de la programación, incluyendo: - Variables y almacenamiento de datos - Operaciones básicas y expresiones - Sentencias condicionales - Primeros pasos para pensar como un programador

Prepárate para resolver problemas emocionantes usando tu nuevo cuaderno de programación y las habilidades que has comenzado a desarrollar.

Capítulo 2: El compilador humano - Entendiendo la lógica y la estructura

Este capítulo introduce conceptos fundamentales de lógica y estructura de programas que forman la base del pensamiento computacional.

Objetivos del capítulo

- Comprender operaciones lógicas básicas y valores booleanos
- Aprender a crear y seguir diagramas de flujo
- Practicar la escritura de pseudocódigo para expresar algoritmos
- Desarrollar habilidades de pensamiento lógico a través de ejercicios de toma de decisiones

Secciones

1. Lógica básica y toma de decisiones
2. Declaraciones condicionales y diagramas de flujo
3. Pseudocódigo

Actividades

- Acertijos lógicos de verdadero/falso
- Creando diagramas de flujo para decisiones cotidianas
- Traduciendo instrucciones en lenguaje natural a pseudocódigo
- La computadora humana: Representando programas simples

Resumen del capítulo

¿Listo para revisar lo que has aprendido? Consulta el Resumen del Capítulo para un repaso de conceptos clave y una vista previa de lo que viene a continuación.

basic logic and decision making

Lógica básica y toma de decisiones

Introducción

En el capítulo anterior, exploramos por qué la programación es importante y cómo puedes aprenderla sin una computadora. Ahora, nos sumergiremos en el corazón de cómo "piensan" las computadoras explorando los fundamentos de la lógica y la toma de decisiones.

Considera este capítulo como un aprendizaje del lenguaje del pensamiento claro—una habilidad que te servirá bien ya sea que estés escribiendo código o tomando decisiones cotidianas.

¿Qué es la lógica?

La lógica es el estudio del razonamiento, particularmente enfocado en cómo determinamos si las afirmaciones son verdaderas o falsas. En nuestra vida cotidiana, usamos la lógica constantemente:

- "Si está lloviendo, debo llevar un paraguas."
- "Como la tienda está cerrada hoy, iré mañana."
- "O tomo el autobús o llegaré tarde a la escuela."

En la programación de computadoras, la lógica funciona de manera similar, pero con reglas muy estrictas. Las computadoras no pueden manejar la ambigüedad con la que los humanos navegan fácilmente. Necesitan instrucciones precisas y claras basadas en si las condiciones son verdaderas o falsas.

Lógica booleana: El fundamento de la computación

En su forma más simple, la lógica informática se basa en un sistema llamado "lógica booleana", nombrado así por el matemático George Boole. Trata solamente con dos valores posibles:

- **Verdadero** (a menudo representado como 1, "sí" o "encendido")
- **Falso** (a menudo representado como 0, "no" o "apagado")

Este enfoque binario puede parecer limitado, pero en realidad es increíblemente poderoso. Las decisiones complejas en la computación se construyen a partir de estos bloques básicos de verdadero/falso.

Valores booleanos en la vida real

Antes de profundizar más, identifiquemos valores booleanos en escenarios cotidianos:

- El interruptor de la luz está encendido (verdadero) o apagado (falso)

- La puerta está abierta (verdadero) o cerrada (falso)
- Tengo suficiente dinero para comprar este artículo (verdadero o falso)
- Actualmente está lloviendo (verdadero o falso)

Actividad: En tu cuaderno, enumera 5 afirmaciones booleanas sobre tu día de hoy—cosas que solo pueden ser verdaderas o falsas.

Operadores booleanos: AND, OR y NOT

Para construir estructuras lógicas más complejas, usamos tres operadores básicos:

1. AND (Conjunción lógica)

El operador AND combina dos valores booleanos y resulta en verdadero SOLO si ambos valores son verdaderos.

Piensa en AND como un amigo exigente que solo está satisfecho cuando todo es perfecto.

Afirmación A	Afirmación B	A AND B
verdadero	verdadero	verdadero
verdadero	falso	falso
falso	verdadero	falso
falso	falso	falso

Ejemplo: “Iré al parque si hace sol Y he terminado mi tarea.” - Sol: verdadero, Tarea terminada: verdadero → Voy al parque (verdadero) - Sol: verdadero, Tarea terminada: falso → No voy al parque (falso) - Sol: falso, Tarea terminada: verdadero → No voy al parque (falso) - Sol: falso, Tarea terminada: falso → No voy al parque (falso)

2. OR (Disyunción lógica)

El operador OR combina dos valores booleanos y resulta en verdadero si al menos uno de los valores es verdadero.

Piensa en OR como un amigo despreocupado que está feliz si ocurre algo bueno.

Afirmación A	Afirmación B	A OR B
verdadero	verdadero	verdadero
verdadero	falso	verdadero
falso	verdadero	verdadero
falso	falso	falso

Ejemplo: “Llevaré un paraguas si está lloviendo O el pronóstico predice lluvia.” - Lloviendo: verdadero, Pronóstico lluvia: verdadero → Llevar paraguas (verdadero) - Lloviendo: verdadero, Pronóstico lluvia: falso → Llevar paraguas (verdadero) - Lloviendo: falso, Pronóstico lluvia: verdadero → Llevar paraguas (verdadero) - Lloviendo: falso, Pronóstico lluvia: falso → No llevar paraguas (falso)

3. NOT (Negación lógica)

El operador NOT simplemente invierte un valor booleano. Si algo es verdadero, NOT lo hace falso, y viceversa.

Piensa en NOT como alguien que siempre contradice lo que dices.

Afirmación A	NOT A
verdadero	falso
falso	verdadero

Ejemplo: “Si NO está lloviendo, iré a caminar.” - Lloviendo: verdadero → NO lloviendo: falso → No voy a caminar - Lloviendo: falso → NO lloviendo: verdadero → Voy a caminar

Tablas de verdad: Mapeando la lógica

Las tablas que hemos estado usando se llaman “tablas de verdad”. Nos ayudan a visualizar todas las posibles combinaciones de entradas y salidas para operaciones lógicas. Las tablas de verdad son especialmente útiles cuando la lógica se vuelve compleja.

Tomando decisiones basadas en la lógica

En programación, la lógica se utiliza para tomar decisiones. Aquí está el patrón general:

```
SI (alguna condición es verdadera) ENTONCES
    (hacer algo)
SI NO
    (hacer otra cosa)
FIN SI
```

Esta estructura aparece en prácticamente todos los lenguajes de programación, aunque la sintaxis exacta puede variar. Es el fundamento de la toma de decisiones en el código.

Toma de decisiones en la vida real

Exploremos una decisión de la vida real a través de la lente de la lógica de programación:

Escenario: Decidir qué vestir según el clima

```
SI (está lloviendo) ENTONCES
    Usar impermeable y llevar paraguas
SI NO
    SI (está soleado Y hace calor) ENTONCES
        Usar ropa ligera y un sombrero
    SI NO
        Usar ropa normal y quizás llevar una chaqueta ligera
    FIN SI
FIN SI
```

Observa cómo podemos anidar decisiones dentro de decisiones para manejar escenarios más complejos.

Combinando múltiples condiciones

La toma de decisiones a menudo involucra múltiples condiciones:

```
SI (es fin de semana) Y (el clima es bueno) ENTONCES
    Ir al parque
SI NO
    Quedarse en casa y leer un libro
FIN SI
```

Usar combinaciones de AND, OR y NOT nos permite crear estructuras de decisión sofisticadas:

```
SI (es un día festivo) O ((es fin de semana) Y (no tengo tarea)) ENTONCES
    Planear algo divertido con amigos
SI NO
    Ponerme al día con los estudios
FIN SI
```

Ejemplo práctico: La decisión de la fiesta

Veamos un ejemplo más complejo:

Escenario: Decidir si ir a una fiesta

Condiciones: - Es en una noche de escuela - Tienes un examen mañana - Tu mejor amigo realmente quiere que vayas - La fiesta está cerca de tu casa

Expresemos esto como una decisión lógica:

```
SI (NO es_noche_escolar) O (NO tengo_examen_mañana Y fiesta_está_cerca) ENTONCES
```

```
    Ir a la fiesta
SI NO
    SI (mejor_amigo_realmente_quiere_que_vayas Y fiesta_está_cerca Y NO tengo_examen_mañana)
        Ir a la fiesta pero salir temprano
    SI NO
        Quedarse en casa
    FIN SI
FIN SI
```

Actividad: Lógica en acción

En tu cuaderno, escribe la lógica para al menos dos decisiones cotidianas que tomas, usando SI, ENTONCES, SI NO, y los operadores booleanos Y, O y NO. Intenta incluir al menos una decisión compleja con múltiples condiciones.

Por ejemplo: - Elegir qué comer para el almuerzo - Decidir si tomar el autobús o caminar - Seleccionar qué materia estudiar primero

Puntos clave

- La lógica booleana utiliza solo dos valores: verdadero y falso
- Los tres operadores booleanos básicos son AND (Y), OR (O) y NOT (NO)
- Las tablas de verdad ayudan a visualizar todos los posibles resultados de las operaciones lógicas
- Las estructuras de decisión en programación se construyen usando patrones SI-ENTONCES-SI NO
- Las decisiones complejas pueden modelarse combinando y anidando estructuras lógicas

En la siguiente sección, construiremos sobre estos fundamentos para explorar las declaraciones condicionales y los diagramas de flujo, que nos darán herramientas poderosas para visualizar y estructurar procesos de toma de decisiones más complejos.

conditional statements and flowcharts

Declaraciones condicionales y diagramas de flujo

Introducción

En la sección anterior, aprendimos sobre la lógica booleana y cómo tomar decisiones básicas usando estructuras SI-ENTONCES-SI NO. Ahora, ampliaremos estos conceptos explorando las declaraciones condicionales con más detalle e introduciendo los diagramas de flujo—herramientas visuales que nos ayudan a mapear la lógica de nuestros programas.

Entendiendo las declaraciones condicionales

Las declaraciones condicionales son la columna vertebral de la toma de decisiones en programación. Permiten que un programa realice diferentes acciones basándose en si ciertas condiciones son verdaderas o falsas.

La estructura básica de una declaración condicional es:

```
SI condición ENTONCES
    hacer algo
SI NO
    hacer otra cosa
FIN SI
```

Examinemos cada parte:

- **SI**: Señala el inicio de una declaración condicional
- **condición**: Una expresión booleana que se evalúa como verdadera o falsa
- **ENTONCES**: Marca lo que sucede si la condición es verdadera
- **hacer algo**: Las acciones que ocurren si la condición es verdadera
- **SI NO**: Introduce las acciones alternativas
- **hacer otra cosa**: Las acciones que ocurren si la condición es falsa
- **FIN SI**: Señala el final de la declaración condicional

Tipos de declaraciones condicionales

1. Declaración SI simple

La forma más simple solo ejecuta código cuando una condición es verdadera:

```
SI está lloviendo ENTONCES
    llevar un paraguas
FIN SI
```

En este caso, no sucede nada específico si no está lloviendo. El programa simplemente continúa con las siguientes instrucciones.

2. Declaración SI-SI NO

Esta forma proporciona dos caminos: uno para cuando la condición es verdadera y otro para cuando es falsa:

```
SI temperatura > 30 grados ENTONCES
    usar ropa ligera
SI NO
    usar una chaqueta
FIN SI
```

3. Declaraciones SI anidadas

Las declaraciones condicionales pueden colocarse dentro de otras declaraciones condicionales para manejar escenarios más complejos:

```
SI es un día de semana ENTONCES
    SI es por la mañana ENTONCES
        ir a la escuela
    SI NO
        hacer la tarea
    FIN SI
SI NO
    relajarse y jugar
FIN SI
```

4. Declaración SI NO SI (o SINO SI)

Cuando necesitamos verificar múltiples condiciones en secuencia:

```
SI puntuación >= 90 ENTONCES
    calificación = "A"
SI NO SI puntuación >= 80 ENTONCES
    calificación = "B"
SI NO SI puntuación >= 70 ENTONCES
    calificación = "C"
SI NO SI puntuación >= 60 ENTONCES
    calificación = "D"
SI NO
    calificación = "F"
FIN SI
```

En este ejemplo, estamos verificando una serie de condiciones en orden, y solo un bloque de código se ejecutará.

Introducción a los diagramas de flujo

Un diagrama de flujo es un diagrama que representa un proceso o flujo de trabajo, mostrando los pasos como cajas de diferentes tipos, y su orden conectándolas

con flechas. Los diagramas de flujo son particularmente útiles para visualizar la lógica de los programas, especialmente aquellos con declaraciones condicionales.

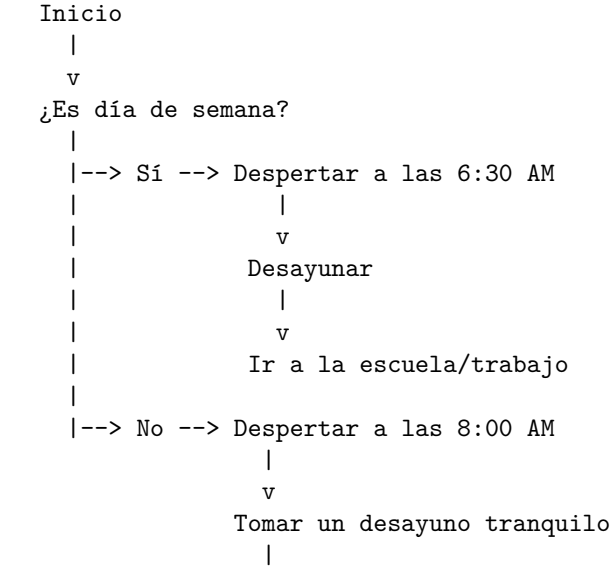
Símbolos básicos de los diagramas de flujo

Estos son los símbolos más comunes utilizados en los diagramas de flujo:

1. **Inicio/Fin (Óvalo o Rectángulo redondeado)**
   - Se utiliza para indicar el comienzo o el final de un proceso
   - Ejemplo: "Inicio" o "Fin"
2. **Proceso (Rectángulo)**
   - Representa un paso en el proceso o una acción a tomar
   - Ejemplo: "Agregar 2 tazas de harina" o "Calcular precio total"
3. **Decisión (Diamante)**
   - Muestra un punto donde se debe tomar una decisión
   - Típicamente contiene una pregunta con una respuesta sí/no o verdadero/falso
   - Ejemplo: "¿Está lloviendo?" o "¿Es x > 10?"
4. **Líneas de flujo (Flechas)**
   - Conectan los símbolos para mostrar la secuencia de pasos
   - Indican la dirección del flujo del proceso
5. **Entrada/Salida (Paralelogramo)**
   - Representa operaciones de entrada o salida
   - Ejemplo: "Ingresa tu nombre" o "Mostrar total"

Creando un diagrama de flujo simple

Vamos a crear un diagrama de flujo para una rutina matutina simple:



```
                v
              Disfrutar tiempo libre
                |
                v
Fin
```

Este diagrama de flujo muestra claramente los diferentes caminos que nuestra mañana podría tomar dependiendo de si es un día de semana o no.

Traduciendo entre declaraciones condicionales y diagramas de flujo

Las dos representaciones—código y diagramas de flujo—pueden traducirse fácilmente entre sí. Por ejemplo, la rutina matutina en código sería:

```
SI es un día de semana ENTONCES
    Despertar a las 6:30 AM
    Desayunar
    Ir a la escuela/trabajo
SI NO
    Despertar a las 8:00 AM
    Tomar un desayuno tranquilo
    Disfrutar tiempo libre
FIN SI
```

La correspondencia entre las dos representaciones es directa e intencional. Los diagramas de flujo proporcionan una visión general visual de la lógica del programa, mientras que el código proporciona las instrucciones detalladas.

Cuándo usar diagramas de flujo

Los diagramas de flujo son particularmente útiles cuando:

1. Planificamos un programa antes de escribir código
2. Explicamos nuestra lógica a otros
3. Depuramos estructuras de decisión complejas
4. Documentamos cómo funciona un programa

Actividad: Toma de decisiones con diagramas de flujo

Practiquemos creando un diagrama de flujo para decidir qué hacer en una tarde de sábado.

Aquí hay un conjunto de reglas: - Si está lloviendo, te quedarás dentro y leerás un libro o verás una película - Si no está lloviendo pero hace mucho calor (más de 35°C), irás a la piscina - Si no está lloviendo y la temperatura es agradable, irás al parque - Si no está lloviendo pero hace frío (menos de 15°C), visitarás la casa de un amigo

Dibuja el diagrama de flujo para este proceso de decisión en tu cuaderno. Asegúrate de usar los símbolos adecuados para inicio/fin, decisiones y procesos.

Condiciones complejas en diagramas de flujo

Los diagramas de flujo también pueden representar condiciones booleanas complejas:

Condición AND (Y)

Cuando se usa AND, ambas condiciones deben ser verdaderas para seguir el camino "Sí":

```
¿Está soleado?
  |
  |--> Sí --> ¿Tengo tiempo libre?
  |               |
  |               |--> Sí --> Ir a la playa
  |               |
  |               |--> No --> Quedarse en casa y mirar por la ventana
  |
  |--> No --> (siguiente decisión)
```

Condición OR (O)

Cuando se usa OR, cualquiera de las condiciones siendo verdadera es suficiente para seguir el camino "Sí":

```
¿Está lloviendo O nevando?
  |
  |--> Sí --> Quedarse en interior
  |
  |--> No --> (siguiente decisión)
```

Decisiones anidadas vs. condiciones compuestas

A menudo hay múltiples formas de representar la misma lógica. Considera estos enfoques equivalentes:

Enfoque 1: Decisiones anidadas

```
¿Es fin de semana?
  |
  |--> Sí --> ¿El clima es bueno?
  |               |
  |               |--> Sí --> Ir al parque
  |               |
  |               |--> No --> Quedarse en casa
```

```
  |
  |--> No --> Quedarse en casa
```

Enfoque 2: Condición compuesta

```
¿Es fin de semana Y el clima es bueno?
  |
  |--> Sí --> Ir al parque
  |
  |--> No --> Quedarse en casa
```

Ambos enfoques conducen a los mismos resultados, pero el segundo es más conciso. A medida que ganes experiencia con la lógica de programación, desarrollarás una intuición para qué representación funciona mejor en diferentes situaciones.

Errores comunes en la lógica condicional

1. Olvidar casos límite

Siempre considera todos los caminos posibles a través de tu lógica. ¿Qué sucede en casos especiales o extremos?

2. Condiciones superpuestas

Ten cuidado cuando las condiciones pueden superponerse. Por ejemplo:

```
SI puntuación > 90 ENTONCES
    calificación = "A"
SI puntuación > 80 ENTONCES
    calificación = "B"
...
```

En este caso, una puntuación de 95 primero establecería la calificación como "A", pero luego inmediatamente la sobrescribiría con "B". El enfoque correcto es usar SI NO SI para hacer que las condiciones sean mutuamente excluyentes.

3. Bucles infinitos

Cuando se usan diagramas de flujo para representar procesos repetitivos (que exploraremos más en capítulos futuros), ten cuidado de no crear caminos que nunca terminen.

Actividad: Diagramando una decisión de la vida real

Piensa en una decisión significativa que tomaste recientemente o necesitas tomar (como elegir un curso para estudiar, decidir sobre una compra o planificar un evento).

1. Enumera todos los factores que influyen en la decisión.
2. Determina cómo estos factores se relacionan entre sí (usando AND, OR, NOT).
3. Dibuja un diagrama de flujo que represente el proceso de decisión.
4. Prueba tu diagrama de flujo con diferentes escenarios para ver si produce los resultados esperados.

Por ejemplo, comprar un nuevo par de zapatos podría involucrar factores como precio, comodidad, estilo y necesidad.

Puntos clave

- Las declaraciones condicionales (SI-ENTONCES-SI NO) permiten a los programas tomar decisiones basadas en condiciones
- Hay varios tipos de declaraciones condicionales: SI simple, SI-SI NO, SI anidado y SI NO SI
- Los diagramas de flujo son representaciones visuales de la lógica del programa usando símbolos estandarizados
- Las decisiones en los diagramas de flujo están representadas por formas de diamante con caminos Sí/No
- Las condiciones complejas usando AND, OR y NOT pueden representarse en diagramas de flujo
- Tanto el código como los diagramas de flujo son herramientas para expresar la misma lógica subyacente

En la siguiente sección, exploraremos el pseudocódigo—una forma de escribir instrucciones similares a programas en una forma que es más fácil de leer y escribir para los humanos, cerrando la brecha entre el lenguaje natural y los lenguajes de programación formales.

pseudo coding

Pseudocódigo

Introducción

En las secciones anteriores, exploramos la lógica booleana, las declaraciones condicionales y los diagramas de flujo. Ahora vamos a aprender sobre el pseudocódigo, una herramienta poderosa que cierra la brecha entre el lenguaje humano y los lenguajes de programación formales.

El pseudocódigo es como un borrador de un programa—expresa la lógica y los pasos en una forma que es más fácil de escribir y entender para los humanos, mientras mantiene suficiente estructura para ser fácilmente traducido a código real más tarde.

¿Qué es el pseudocódigo?

El pseudocódigo es una forma de describir un algoritmo o programa usando una mezcla de lenguaje natural (como el español) y estructuras similares a la programación. No está destinado a ser ejecutado por una computadora sino más bien a ayudar a los programadores a planificar su código y comunicar sus ideas a otros.

Piensa en el pseudocódigo como un conjunto de instrucciones de cocina. Cuando lees una receta, tiene un formato específico y usa ciertas convenciones, pero está escrita de una manera que los humanos pueden entender fácilmente. De manera similar, el pseudocódigo utiliza conceptos de programación pero los expresa en una forma más legible.

¿Por qué usar pseudocódigo?

El pseudocódigo ofrece varias ventajas:

1. **Enfoque en la lógica**: Te permite concentrarte en la lógica de resolución de problemas sin quedar atrapado en detalles de sintaxis del lenguaje de programación.
2. **Comunicación**: Es más fácil de entender para otros (incluso no programadores), lo que lo hace ideal para discutir algoritmos y soluciones.
3. **Planificación**: Te ayuda a organizar tus pensamientos y planificar tu programa antes de escribir código real.
4. **Independencia del lenguaje**: El pseudocódigo no está ligado a ningún lenguaje de programación específico, por lo que el mismo pseudocódigo puede traducirse a diferentes lenguajes.
5. **Prevención de errores**: Al planificar tu lógica en pseudocódigo primero, puedes detectar errores lógicos temprano, antes de escribir código real.

Convenciones del pseudocódigo

Aunque no existe una sintaxis "oficial" única para el pseudocódigo, comúnmente se utilizan ciertas convenciones:

1. **Usar declaraciones descriptivas en español** para la mayoría de las instrucciones
2. **ESCRIBIR EN MAYÚSCULAS** palabras clave como SI, SI NO, MIENTRAS, PARA, etc.
3. **Indentar** bloques de código para mostrar la estructura
4. **Usar símbolos estándar** para operaciones:
   - = para asignación
   - ==, >, <, >=, <= para comparaciones
   - +, -, *, / para operaciones aritméticas
5. **Numerar líneas** (opcional) para facilitar la discusión

Veamos un ejemplo de pseudocódigo para determinar el mayor de tres números:

```
1. INICIO
2. OBTENER numero1, numero2, numero3
3. ESTABLECER mayor = numero1
4. SI numero2 > mayor ENTONCES
5.     ESTABLECER mayor = numero2
6. FIN SI
7. SI numero3 > mayor ENTONCES
8.     ESTABLECER mayor = numero3
9. FIN SI
10. MOSTRAR "El número mayor es " + mayor
11. FIN
```

De diagramas de flujo a pseudocódigo

Una de las fortalezas del pseudocódigo es lo bien que se combina con los diagramas de flujo. Tomemos el diagrama de flujo de actividad de fin de semana de la sección anterior y convirtámoslo a pseudocódigo:

Diagrama de flujo (conceptual):

```
¿Es fin de semana Y el clima es bueno?
  |
  |--> Sí --> Ir al parque
  |
  |--> No --> Quedarse en casa
```

Pseudocódigo:

```
1. OBTENER dia_de_semana
```

```
2. OBTENER condicion_climatica
3. SI dia_de_semana == "Sábado" O dia_de_semana == "Domingo" ENTONCES
4.     SI condicion_climatica == "buena" ENTONCES
5.         MOSTRAR "Ir al parque"
6.     SI NO
7.         MOSTRAR "Quedarse en casa"
8.     FIN SI
9. SI NO
10.    MOSTRAR "Quedarse en casa"
11. FIN SI
```

Observa cómo el pseudocódigo es más detallado que el diagrama de flujo pero sigue siendo más fácil de leer que el código de programación real.

Elementos comunes del pseudocódigo

Entrada y salida

```
OBTENER nombre_variable      // Para entrada
MOSTRAR mensaje              // Para salida
```

Variables y asignación

```
ESTABLECER variable = valor  // Asigna un valor a una variable
```

Declaraciones condicionales

```
SI condición ENTONCES        // SI simple
    declaraciones
FIN SI

SI condición ENTONCES        // SI-SI NO
    declaraciones1
SI NO
    declaraciones2
FIN SI

SI condición1 ENTONCES       // SI-SI NO SI-SI NO
    declaraciones1
SI NO SI condición2 ENTONCES
    declaraciones2
SI NO
    declaraciones3
FIN SI
```

Bucles (que exploraremos más en capítulos posteriores)

```
MIENTRAS condición HACER     // Bucle MIENTRAS
```

```
    declaraciones
FIN MIENTRAS

PARA i = inicio HASTA fin    // Bucle PARA
    declaraciones
FIN PARA
```

Funciones (que también exploraremos más adelante)

```
FUNCIÓN nombre(parámetros)
    declaraciones
    DEVOLVER valor
FIN FUNCIÓN
```

Ejemplo: Usando pseudocódigo para planificar una solución

Usemos pseudocódigo para planificar una solución para un problema común: determinar si un año es bisiesto.

Un año bisiesto es un año que es divisible por 4, excepto los años que son divisibles por 100 pero no por 400.

Aquí está el pseudocódigo:

```
1. INICIO
2. OBTENER año
3. SI (año es divisible por 400) ENTONCES
4.     ESTABLECER es_bisiesto = verdadero
5. SI NO SI (año es divisible por 100) ENTONCES
6.     ESTABLECER es_bisiesto = falso
7. SI NO SI (año es divisible por 4) ENTONCES
8.     ESTABLECER es_bisiesto = verdadero
9. SI NO
10.    ESTABLECER es_bisiesto = falso
11. FIN SI
12. SI es_bisiesto ENTONCES
13.    MOSTRAR año + " es un año bisiesto"
14. SI NO
15.    MOSTRAR año + " no es un año bisiesto"
16. FIN SI
17. FIN
```

Escribir la lógica en pseudocódigo nos ayuda a detectar posibles problemas antes de comenzar a codificar. Por ejemplo, el orden de las condiciones es crucial; si verificáramos primero la divisibilidad por 4, clasificaríamos incorrectamente años como 1900 (que es divisible por 100 pero no por 400) como años bisiestos.

Traduciendo lenguaje natural a pseudocódigo

A menudo, necesitarás traducir un problema descrito en lenguaje natural a pseudocódigo. Aquí hay un proceso para hacer esto:

1. **Identificar las entradas y salidas**
2. **Descomponer el problema en pasos**
3. **Identificar puntos de decisión**
4. **Escribir pseudocódigo para cada paso**
5. **Revisar y refinar tu solución**

Practiquemos con un ejemplo:

Problema: Un profesor quiere calcular el promedio de las calificaciones de los exámenes de un estudiante, pero quiere eliminar la calificación más baja si el estudiante ha realizado más de tres exámenes.

Paso 1: Identificar entradas y salidas - Entradas: Una lista de calificaciones de exámenes - Salida: El promedio (potencialmente con la calificación más baja eliminada)

Pasos 2-5: Descomponer el problema y escribir pseudocódigo

```
1. INICIO
2. OBTENER calificaciones_examenes (una lista de números)
3. ESTABLECER total = 0
4. ESTABLECER cantidad = número de calificaciones en calificaciones_examenes
5. SI cantidad > 3 ENTONCES
6.     ESTABLECER calificacion_min = primera calificación en calificaciones_examenes
7.     PARA cada calificación en calificaciones_examenes
8.         SI calificación < calificacion_min ENTONCES
9.             ESTABLECER calificacion_min = calificación
10.        FIN SI
11.    FIN PARA
12.    ESTABLECER total = suma de todas las calificaciones en calificaciones_examenes - cali
13.    ESTABLECER cantidad = cantidad - 1
14. SI NO
15.    ESTABLECER total = suma de todas las calificaciones en calificaciones_examenes
16. FIN SI
17. ESTABLECER promedio = total / cantidad
18. MOSTRAR "El promedio es " + promedio
19. FIN
```

Actividad: Traduciendo problemas a pseudocódigo

Intenta convertir estos problemas del mundo real a pseudocódigo:

1. **Problema**: Determinar si un estudiante ha aprobado un curso. Para aprobar, el estudiante debe tener un promedio de al menos 60% y haber asistido al menos al 80% de las clases.

2. **Problema**: Calcular el costo de un viaje en taxi. La tarifa base es $2.50, y luego $0.50 por kilómetro. Si la distancia total es más de 10 kilómetros, se aplica un descuento del 5% a la tarifa total.

3. **Problema**: Una máquina expendedora da cambio usando la menor cantidad de monedas posible. Dado un monto de cambio a devolver, determinar cuántas monedas de 25¢, 10¢, 5¢, y 1¢ proporcionar.

Después de escribir tu pseudocódigo, pruébalo con ejemplos específicos para asegurarte de que funciona correctamente.

Mejores prácticas para el pseudocódigo

Para escribir pseudocódigo efectivo:

1. **Sé claro y conciso**: Usa lenguaje simple que cualquiera pueda entender.
2. **Sé consistente**: Elige un estilo y mantente con él a lo largo de tu pseudocódigo.
3. **Usa el nivel adecuado de detalle**: Incluye suficiente detalle para entender la lógica, pero no te pierdas en detalles específicos de implementación.
4. **Piensa paso a paso**: Descompón operaciones complejas en pasos más simples.
5. **Usa nombres de variables significativos**: Elige nombres que describan lo que representan las variables.
6. **Comenta tu pseudocódigo**: Agrega explicaciones para partes complejas o no obvias.

Del pseudocódigo al código

Una vez que hayas refinado tu pseudocódigo, traducirlo a código real se vuelve mucho más fácil. Aquí hay un ejemplo simple que muestra pseudocódigo y su traducción a varios lenguajes de programación:

Pseudocódigo:

```
SI temperatura > 30 ENTONCES
    MOSTRAR "¡Hace calor!"
SI NO
    MOSTRAR "No hace demasiado calor."
FIN SI
```

Python:

```python
if temperatura > 30:
    print("¡Hace calor!")
else:
    print("No hace demasiado calor.")
```

JavaScript:

```
if (temperatura > 30) {
    console.log("¡Hace calor!");
} else {
    console.log("No hace demasiado calor.");
}
```

Cuando eventualmente comiences a escribir en un lenguaje de programación específico, encontrarás que la transición es mucho más fluida si ya has elaborado la lógica en pseudocódigo.

Actividad: Implementando pseudocódigo en la vida real

¡El pseudocódigo no es solo para programas de computadora—puede ayudar con procesos de la vida real también!

1. Elige una tarea rutinaria que realizas regularmente (como preparar el desayuno, alistarte para la escuela, u organizar tu tiempo de estudio).
2. Escribe pseudocódigo para este proceso, incluyendo puntos de decisión y acciones repetitivas.
3. Prueba tu pseudocódigo siguiéndolo paso a paso.
4. Revisa tu pseudocódigo para hacer el proceso más eficiente.

Este ejercicio ayuda a desarrollar el pensamiento algorítmico para situaciones cotidianas.

Puntos clave

- El pseudocódigo es una forma de describir algoritmos usando una mezcla de lenguaje natural y estructuras similares a la programación
- Cierra la brecha entre el pensamiento humano y los lenguajes de programación formales
- El pseudocódigo ayuda a enfocarse en la lógica de una solución sin quedar atrapado en la sintaxis específica del lenguaje
- Aunque no hay un estándar único para el pseudocódigo, la consistencia y la claridad son importantes
- El pseudocódigo funciona bien con los diagramas de flujo—se complementan mutuamente
- Desarrollar habilidades sólidas de pseudocódigo facilita la transición a lenguajes de programación reales

En este capítulo, hemos construido una base sólida en pensamiento lógico y estructura de programa. Hemos explorado la lógica booleana y las tablas de verdad, las declaraciones condicionales y los diagramas de flujo, y finalmente el pseudocódigo como un puente para expresar algoritmos de manera más formal. Estos bloques de construcción son esenciales para la programación y el

pensamiento computacional, y te servirán bien a medida que profundizamos en conceptos más complejos en los próximos capítulos.

Activities

truth tables and logic puzzles

Actividad: Tablas de verdad y acertijos lógicos

Visión general

Esta actividad proporciona práctica práctica con lógica booleana utilizando tablas de verdad y acertijos lógicos. Al trabajar en estos ejercicios, desarrollarás tu comprensión de cómo funcionan los operadores lógicos (AND, OR, NOT) y cómo pueden combinarse para expresar condiciones complejas.

Objetivos de aprendizaje

- Crear e interpretar tablas de verdad para operaciones lógicas básicas
- Evaluar expresiones lógicas complejas
- Traducir escenarios cotidianos a expresiones lógicas
- Identificar patrones en operaciones lógicas
- Desarrollar habilidades de razonamiento lógico esenciales para la programación

Materiales necesarios

- Tu cuaderno
- Lápiz (y borrador)
- Opcional: Regla para dibujar tablas

Tiempo requerido

45-60 minutos

Parte 1: Creando tablas de verdad básicas

Paso 1: Configura tablas de verdad para operaciones básicas

En tu cuaderno, crea tres tablas de verdad separadas para las operaciones lógicas básicas: AND, OR y NOT.

Para AND y OR, configura tus tablas con tres columnas: - A - B - Resultado (A AND B) o (A OR B)

Para NOT, solo necesitarás dos columnas: - A - Resultado (NOT A)

Paso 2: Completa los valores de verdad

Completa todas las combinaciones posibles de valores VERDADERO y FALSO para cada tabla y determina los resultados basándote en las reglas:

- AND: Devuelve VERDADERO solo cuando ambas entradas son VERDADERO

- OR: Devuelve VERDADERO cuando al menos una entrada es VERDADERO
- NOT: Devuelve lo opuesto del valor de entrada

Ejemplo:

Para la operación AND, tu tabla completada debería verse así:

A	B	A AND B
VERDADERO	VERDADERO	VERDADERO
VERDADERO	FALSO	FALSO
FALSO	VERDADERO	FALSO
FALSO	FALSO	FALSO

Parte 2: Expresiones lógicas compuestas

Ahora practiquemos con expresiones más complejas que combinan múltiples operadores.

Paso 1: Configura tablas de verdad para expresiones compuestas

Crea tablas de verdad para cada una de las siguientes expresiones:

1. (A AND B) OR C
2. A AND (B OR C)
3. NOT (A AND B)
4. (NOT A) OR (NOT B)

Para cada expresión, tu tabla necesitará cuatro columnas: - A - B - C - Resultado

Paso 2: Completa todas las combinaciones posibles

Cada tabla tendrá 8 filas ($2^3 = 8$ combinaciones posibles con tres variables).

Paso 3: Evalúa paso a paso

Para cada expresión, trabaja la lógica paso a paso. Por ejemplo, para "(A AND B) OR C": 1. Primero calcula (A AND B) 2. Luego calcula el resultado final: (A AND B) OR C

Ejemplo:

Para la expresión (A AND B) OR C, las primeras filas podrían verse así:

A	B	C	(A AND B)	(A AND B) OR C
VERDADERO	VERDADERO	VERDADERO	VERDADERO	VERDADERO

A	B	C	(A AND B)	(A AND B) OR C
VERDADERO	VERDADERO	FALSO	VERDADERO	VERDADERO
VERDADERO	FALSO	VERDADERO	FALSO	VERDADERO
…	…	…	…	…

Puedes crear una columna intermedia como se muestra para ayudar con los cálculos, o simplemente resolverlo paso a paso en tu mente.

Parte 3: Equivalencias lógicas

Algunas expresiones lógicas diferentes pueden ser equivalentes—siempre producen los mismos resultados para las mismas entradas.

Paso 1: Compara tus tablas de verdad

Observa tus tablas de verdad completadas para las expresiones 3 y 4: - NOT (A AND B) - (NOT A) OR (NOT B)

Compara las columnas de resultados. ¿Son iguales para todas las combinaciones de entrada?

Paso 2: Descubre las Leyes de De Morgan

Lo que acabas de verificar es una de las Leyes de De Morgan, un principio importante en lógica: - NOT (A AND B) es equivalente a (NOT A) OR (NOT B)

La otra Ley de De Morgan establece: - NOT (A OR B) es equivalente a (NOT A) AND (NOT B)

Paso 3: Verifica la segunda ley

Crea tablas de verdad para verificar la segunda Ley de De Morgan.

Parte 4: Acertijos lógicos

¡Ahora apliquemos la lógica booleana para resolver algunos acertijos!

Acertijo 1: Detectando mentiras

Tres personas (Ali, Bo y Cal) hacen una declaración cada una, pero sabes que solo una de ellas está diciendo la verdad.

- Ali dice: “Estoy diciendo la verdad.”
- Bo dice: “Ali está mintiendo.”
- Cal dice: “Bo está mintiendo.”

¿Quién está diciendo la verdad?

Para resolver esto, crea una tabla de verdad con todas las posibilidades (cada persona dice la verdad T o miente M), y verifica qué escenario coincide con la condición de que exactamente una persona dice la verdad.

Acertijo 2: Los interruptores de luz

Estás en una habitación con tres interruptores de luz, cada uno conectado a una bombilla diferente en otra habitación. No puedes ver las bombillas desde donde están los interruptores, y solo puedes ir a la otra habitación una vez para revisar las bombillas.

¿Cómo puedes determinar qué interruptor controla qué bombilla?

Piensa en este acertijo en términos de los estados (ENCENDIDO/APAGADO) y qué información puedes reunir con solo una visita a la otra habitación.

Acertijo 3: Deducción lógica

Basándote en estas pistas, determina quién tiene qué mascota: - Hay tres amigos: Xia, Yoon y Zach - Cada uno tiene una mascota: un perro, un gato o un pájaro - Xia no tiene el pájaro - Si Yoon tiene el gato, entonces Zach tiene el pájaro - Si Zach no tiene el perro, entonces Xia tiene el perro

Crea una tabla para rastrear posibilidades y usa la lógica booleana para reducir la respuesta.

Parte 5: Aplicaciones del mundo real

Aplicación 1: Criterios de elegibilidad

Una beca tiene los siguientes requisitos de elegibilidad: - El estudiante debe tener un promedio de al menos 3.5 O - El estudiante debe haber completado al menos 30 horas de servicio comunitario Y tener un promedio de al menos 3.0

Escribe esto como una expresión lógica usando variables: - Sea G = “El promedio es al menos 3.5” - Sea H = “Completó al menos 30 horas de servicio comunitario” - Sea M = “El promedio es al menos 3.0”

Luego crea una tabla de verdad para mostrar todas las combinaciones de estas variables y si cada combinación calificaría para la beca.

Aplicación 2: Personalización de menú

El sistema de pedidos de un restaurante utiliza lógica para determinar combinaciones de comidas: - Cada comida viene con arroz O papas (pero no ambos) - Si eliges el plato de pescado, debes tener verduras - Si eliges verduras, puedes tener salsa A O salsa B (pero no ambas)

Crea variables para cada elección y escribe expresiones lógicas para combinaciones válidas de comidas.

Actividades de extensión

1. Crea tu propio acertijo lógico

Diseña un acertijo lógico similar a los de la Parte 4, y proporciona su solución. Intercambia acertijos con compañeros de clase si es posible.

2. Explora NAND y NOR

Investiga dos operaciones lógicas adicionales: NAND (NOT AND) y NOR (NOT OR). Crea sus tablas de verdad y explora cómo cualquier expresión lógica puede construirse usando solo operaciones NAND o solo operaciones NOR.

3. Diagramas de Venn

Dibuja diagramas de Venn para representar las operaciones AND, OR y NOT visualmente. Luego usa diagramas de Venn para ilustrar las expresiones compuestas de la Parte 2.

Preguntas de reflexión

En tu cuaderno, responde estas preguntas: 1. ¿Qué operación lógica (AND, OR, NOT) fue más fácil de entender para ti? ¿Cuál fue la más desafiante? 2. ¿Cómo ayudan las Leyes de De Morgan a simplificar expresiones lógicas complejas? 3. ¿Cómo podrías usar la lógica booleana en tu toma de decisiones cotidiana? 4. ¿Qué patrones notaste en las tablas de verdad que creaste? 5. ¿Cómo crees que las computadoras usan estas operaciones lógicas para tomar decisiones?

Conexión con la programación

La lógica booleana que has practicado en esta actividad forma la base de la toma de decisiones en programación. Cada declaración condicional (IF-THEN-ELSE) se basa en evaluar expresiones lógicas. Entender estos principios te ayudará a escribir código claro y efectivo y a depurar errores lógicos en tus programas cuando eventualmente comiences a programar en una computadora.

creating flowcharts

Actividad: Creando diagramas de flujo para decisiones cotidianas

Visión general

Esta actividad te ayuda a practicar la creación de diagramas de flujo para visualizar procesos de toma de decisiones. Aprenderás a mapear pasos lógicos para escenarios cotidianos, utilizando símbolos estándar de diagramas de flujo para representar diferentes tipos de operaciones.

Objetivos de aprendizaje

- Utilizar los símbolos apropiados de diagramas de flujo para representar diferentes operaciones
- Crear flujos claros y lógicos para procesos de toma de decisiones
- Traducir escenarios del mundo real en diagramas de flujo estructurados
- Identificar puntos de decisión y sus posibles resultados
- Desarrollar habilidades de pensamiento visual que complementen el razonamiento lógico

Materiales necesarios

- Tu cuaderno (preferiblemente con páginas en blanco o cuadriculadas)
- Lápiz y borrador
- Regla (opcional pero útil)
- Lápices o bolígrafos de colores (opcional)
- Plantillas de símbolos de diagramas de flujo (proporcionadas en esta actividad)

Tiempo requerido

45-60 minutos

Parte 1: Símbolos y convenciones de los diagramas de flujo

Símbolos estándar de los diagramas de flujo

En tu cuaderno, crea una página de referencia con estos símbolos estándar de diagramas de flujo:

1. **Inicio/Fin (Óvalo)**
   - Propósito: Marca el comienzo o el final de un proceso
   - Texto de ejemplo: “Inicio” o “Fin”
2. **Proceso (Rectángulo)**
   - Propósito: Representa una acción u operación

- Texto de ejemplo: “Agregar azúcar a la mezcla” o “Calcular precio total”

3. **Decisión (Diamante)**
   - Propósito: Indica un punto de decisión con múltiples caminos
   - Texto de ejemplo: “¿Está lloviendo?” o “¿Temperatura > 30°C?”
4. **Entrada/Salida (Paralelogramo)**
   - Propósito: Muestra entrada o salida de datos
   - Texto de ejemplo: “Ingresa tu nombre” o “Mostrar costo total”
5. **Líneas de flujo (Flechas)**
   - Propósito: Conectan símbolos y muestran la secuencia del flujo
   - Ejemplo: Flechas simples con indicadores de dirección
6. **Conector (Círculo)**
   - Propósito: Conecta diferentes partes de un diagrama de flujo, especialmente entre páginas
   - Texto de ejemplo: “A” o “1”

Convenciones de los diagramas de flujo

Junto a tus símbolos, anota estas importantes convenciones:

1. El flujo generalmente se mueve de arriba hacia abajo y de izquierda a derecha
2. Los diamantes de decisión tienen exactamente dos salidas (generalmente etiquetadas “Sí/No” o “Verdadero/Falso”)
3. Cada camino debe eventualmente llegar a un punto final
4. Las líneas no deben cruzarse si es posible (usa conectores si es necesario)
5. Usa espaciado y tamaño consistente de símbolos

Parte 2: Práctica de diagramas de flujo simples

Paso 1: Crea un diagrama de flujo de una rutina matutina

Dibuja un diagrama de flujo para una rutina matutina básica con estos elementos: - Comienza cuando te despiertas - Decisión sobre si es día de semana o fin de semana - Diferentes rutinas basadas en el tipo de día - Termina cuando sales de casa o comienzas las actividades de tu día

Paso 2: Agrega una decisión sobre el clima

Mejora tu diagrama de flujo para incluir una decisión relacionada con el clima: - Verifica si está lloviendo - Si sí, lleva un paraguas - Si no, procede normalmente

Paso 3: Revisa y refina

Verifica tu diagrama de flujo para: - Uso adecuado de símbolos - Dirección clara del flujo - Progresión lógica - Caminos completos (sin callejones sin salida)

Parte 3: Diagramas de flujo para decisiones cotidianas

Ahora crearás diagramas de flujo para escenarios cotidianos más complejos. Elige dos de los siguientes escenarios para crear diagramas de flujo completos:

Escenario 1: Decidir qué comer para la cena

Considera factores como: - ¿Qué ingredientes están disponibles? - ¿Cuánto tiempo tienes? - ¿Estás cocinando para ti o para otros? - ¿Tienes restricciones dietéticas?

Escenario 2: Planificar una actividad de fin de semana

Considera factores como: - Condiciones climáticas - Restricciones de presupuesto - Quién te acompañará - Opciones de transporte - Disponibilidad de tiempo

Escenario 3: Elegir un regalo para alguien

Considera factores como: - ¿Cuál es tu presupuesto? - ¿Cuáles son sus intereses? - ¿Ya tienen algo similar? - ¿Es una ocasión especial? - ¿Necesita ser entregado?

Escenario 4: Solución de problemas de un dispositivo que no funciona

Crea un diagrama de flujo para diagnosticar por qué un dispositivo (como un teléfono móvil) no está funcionando, con pasos como: - ¿Está encendido? - ¿La batería está cargada? - ¿Todas las conexiones están seguras? - ¿Se ha caído o dañado? - ¿Qué mensajes de error se muestran?

Parte 4: Traduciendo historias a diagramas de flujo

Paso 1: Lee la historia

Lee este breve escenario:

> María quiere hacer un pastel para el cumpleaños de su amiga. Verifica si tiene todos los ingredientes. Si tiene todo, comienza a hornear inmediatamente. Si le falta algo, verifica si la tienda está abierta. Si la tienda está abierta, va a comprar los ingredientes que faltan y luego hornea el pastel. Si la tienda está cerrada, decide hacer galletas en su lugar, siempre y cuando tenga esos ingredientes. Si tampoco tiene los ingredientes para galletas, comprará un pastel de la panadería mañana.

Paso 2: Identifica los elementos clave

En tu cuaderno, enumera: - Todos los puntos de decisión - Todas las acciones posibles - El flujo lógico entre decisiones y acciones

Paso 3: Crea el diagrama de flujo

Dibuja un diagrama de flujo completo que represente el proceso de decisión de María para hornear el pastel.

Paso 4: Prueba tu diagrama de flujo

Recorre tu diagrama de flujo con diferentes escenarios: - María tiene todos los ingredientes del pastel - A María le faltan huevos, la tienda está abierta - A María le falta harina, la tienda está cerrada, pero tiene ingredientes para galletas - A María le falta harina, la tienda está cerrada, y no tiene ingredientes para galletas

Parte 5: Diagramas de flujo para algoritmos

Ahora practiquemos la creación de diagramas de flujo para algoritmos simples (procedimientos paso a paso).

Algoritmo 1: Encontrar el número más grande

Crea un diagrama de flujo para encontrar el mayor de tres números: 1. Ingresa tres números: A, B y C 2. Compara A con B para encontrar el mayor 3. Compara ese resultado con C 4. Muestra el número más grande

Algoritmo 2: Calculando un precio con descuento

Crea un diagrama de flujo para calcular un precio con descuento: 1. Ingresa el precio original 2. Ingresa el porcentaje de descuento 3. Si el precio original es superior a $100, aplica el descuento 4. Si el precio original es $100 o menos, aplica la mitad del descuento 5. Calcula y muestra el precio final

Parte 6: Diagramas de flujo colaborativos (actividad grupal opcional)

Si estás trabajando con otros, prueba este ejercicio colaborativo:

Paso 1: Selecciona un proceso complejo

Elige un proceso complejo que todos conozcan, como: - Tomar una decisión grupal sobre dónde ir a almorzar - Planificar un proyecto o evento de clase - Crear un horario de estudio para exámenes

Paso 2: Borradores individuales

Cada persona crea su propio diagrama de flujo para el proceso.

Paso 3: Comparar y combinar

Compara los diferentes diagramas de flujo y discute: - ¿Qué puntos de decisión incluyó cada uno? - ¿Qué enfoques diferentes se tomaron? - ¿Qué factores importantes consideraron algunos que otros pasaron por alto?

Paso 4: Crear un diagrama de flujo maestro

Trabajen juntos para crear un diagrama de flujo integral que incorpore los mejores elementos de cada diagrama individual.

Plantillas para símbolos de diagramas de flujo

Si te resulta difícil dibujar los símbolos a mano alzada, aquí hay plantillas simplificadas que puedes trazar o copiar:

```
Inicio/Fin:
   .--------.
  /          \
 (            )
  \          /
   '--------'

Proceso:
.--------------.
|              |
|              |
'--------------'

Decisión:
      ./.\.
     /     \
    /       \
   <         >
    \       /
     \     /
      '\.'

Entrada/Salida:
    .--------.
   /          \
  /            \
'--------------'

Conector:
    .---.
   |     |
```

Actividades de extensión

1. Revisión del diagrama de flujo

Toma uno de tus diagramas de flujo y optimízalo: - Reduciendo el número de puntos de decisión si es posible - Encontrando caminos más eficientes - Añadiendo factores adicionales para hacerlo más completo

2. Conceptos de programación en diagramas de flujo

Investiga y crea diagramas de flujo que representen estos conceptos de programación: - Un bucle de conteo simple (1 a 10) - Un proceso de validación de entrada - Un algoritmo de búsqueda simple

3. Diagramas de flujo digitales

Si tienes acceso a una computadora, prueba una de estas herramientas gratuitas de diagramas de flujo en línea: - draw.io - Lucidchart (versión gratuita) - Google Drawings

Preguntas de reflexión

En tu cuaderno, responde estas preguntas: 1. ¿Cómo te ayudó la creación de diagramas de flujo a entender mejor los procesos de decisión? 2. ¿Cuál fue la parte más desafiante de crear diagramas de flujo? 3. ¿Cómo podrían ser útiles los diagramas de flujo en tu vida diaria o estudios? 4. ¿Qué diferencias notaste entre diagramar procesos simples versus complejos? 5. ¿Cómo crees que los diagramas de flujo ayudan a los programadores a planificar su código?

translating to pseudocode

Actividad: Traduciendo del lenguaje natural al pseudocódigo

Visión general

Esta actividad te ayuda a desarrollar la habilidad de traducir instrucciones en lenguaje natural al pseudocódigo. Aprenderás a tomar procesos cotidianos y reescribirlos en un formato más estructurado y paso a paso que cierra la brecha entre el lenguaje humano y el código de programación formal.

Objetivos de aprendizaje

- Convertir descripciones en lenguaje natural en pseudocódigo claro y estructurado
- Practicar el uso de convenciones y sintaxis estándar de pseudocódigo
- Desglosar procesos complejos en pasos lógicos
- Identificar puntos de decisión y estructuras de control dentro de los procesos
- Desarrollar precisión al expresar algoritmos

Materiales necesarios

- Tu cuaderno
- Lápiz y borrador
- La guía de referencia de pseudocódigo (de la Sección 3)

Tiempo requerido

45-60 minutos

Parte 1: Revisión de convenciones de pseudocódigo

Antes de comenzar los ejercicios de traducción, repasemos las convenciones clave del pseudocódigo:

1. **Palabras clave**: Comúnmente en mayúsculas (SI, SI NO, MIENTRAS, PARA, etc.)
2. **Indentación**: Usada para mostrar anidamiento y estructura
3. **Asignación**: Usa = (ej., ESTABLECER total = 0)
4. **Comparación**: Usa ==, >, <, >=, <= (ej., SI edad >= 18 ENTONCES)
5. **Operaciones básicas**:
    - Entrada: OBTENER o ENTRADA
    - Salida: MOSTRAR o SALIDA
    - Procesamiento: ESTABLECER, CALCULAR

- Toma de decisiones: SI-ENTONCES-SI NO
- Repetición: MIENTRAS, PARA
- Función: FUNCIÓN, PROCEDIMIENTO, DEVOLVER

En tu cuaderno, crea una página de referencia con estas convenciones y añade ejemplos de cada una.

Parte 2: Traducciones simples

Paso 1: Estudia el ejemplo

Aquí hay un ejemplo de traducción de una descripción simple en lenguaje natural a pseudocódigo:

Lenguaje natural: Para hacer una taza de té, primero hierve agua en una tetera. Una vez que el agua esté hirviendo, viértela en una taza con una bolsa de té. Déjala reposar por 3 minutos, luego retira la bolsa de té. Añade azúcar si lo deseas.

Pseudocódigo:

```
INICIO
    Hervir agua en tetera
    MIENTRAS el agua no esté hirviendo
        Esperar
    FIN MIENTRAS
    Verter agua en taza con bolsa de té
    Esperar por 3 minutos
    Retirar bolsa de té
    MOSTRAR "¿Quieres azúcar?"
    OBTENER desea_azucar
    SI desea_azucar == "sí" ENTONCES
        Añadir azúcar al té
    FIN SI
FIN
```

Paso 2: Practica con procesos simples

Traduce cada una de estas descripciones en lenguaje natural a pseudocódigo:

1. **Verificar si un número es par o impar**: "Toma un número. Si puedes dividirlo por 2 sin un resto, es par. De lo contrario, es impar."
2. **Hacer un sándwich**: "Toma dos rebanadas de pan. Unta mantequilla en un lado de cada rebanada. Añade queso y jamón entre las rebanadas, con los lados con mantequilla hacia adentro. Opcionalmente, tuesta el sándwich hasta que el queso se derrita."
3. **Configurar una alarma**: "Decide a qué hora necesitas despertar. Resta el tiempo que necesitas para alistarte. Configura tu alarma para esa hora.

Asegúrate de que la alarma esté encendida antes de ir a dormir."

Paso 3: Revisa y refina

Para cada una de tus traducciones: - Verifica que hayas usado las convenciones correctas de pseudocódigo - Asegúrate de que todos los pasos estén incluidos y en el orden correcto - Busca ambigüedades o detalles faltantes en tu pseudocódigo - Asegúrate de que los puntos de decisión (declaraciones SI) tengan condiciones claras

Parte 3: Traduciendo procesos complejos

Ahora abordemos procesos más complejos que involucran múltiples decisiones y posibles bucles.

Paso 1: Ejercicio de traducción

Traduce estos procesos más complejos a pseudocódigo:

1. **Encontrar el máximo de tres números**: "Dados tres números, primero compara los dos primeros números para encontrar cuál es mayor. Luego compara ese resultado con el tercer número para encontrar el mayor de los tres."
2. **Calculando una cuenta de restaurante con propina**: "Suma el costo de todos los platos ordenados. Verifica si ya se incluye un cargo por servicio. Si no, calcula una propina del 15% por buen servicio o 20% por servicio excelente. Añade la propina al total de la cuenta. Divide el total equitativamente entre todos los comensales."
3. **Planificando un viaje**: "Decide un destino. Verifica si tienes suficiente dinero para el viaje. Si lo tienes, reserva transporte y alojamiento. Si no, elige un destino más económico o ahorra más dinero antes de reservar. Antes del viaje, haz una lista de equipaje y empaca tus maletas. El día del viaje, verifica nuevamente que tengas todos los elementos esenciales."

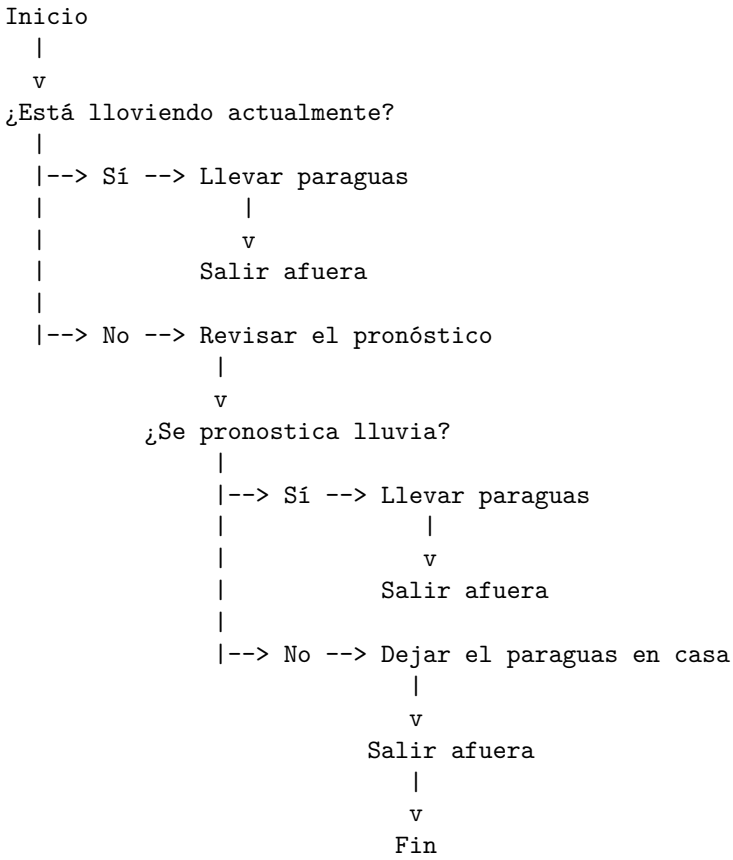
Paso 2: Añade detalle y claridad

Revisa tu pseudocódigo y mejóralo: - Añade comentarios para explicar partes complejas - Usa nombres de variables más específicos (ej., `costo_total` en lugar de solo `total`) - Desglosa pasos muy complejos en más simples - Asegúrate de que todos los casos límite estén manejados

Parte 4: De diagramas de flujo a pseudocódigo

Paso 1: Elige un diagrama de flujo

Selecciona uno de los diagramas de flujo que creaste en la actividad anterior, o usa este ejemplo simple de decidir si llevar un paraguas:



Paso 2: Convierte a pseudocódigo

Traduce el diagrama de flujo seleccionado a pseudocódigo. Recuerda que: - Los diamantes (símbolos de decisión) se convierten en declaraciones SI - Los rectángulos (símbolos de proceso) se convierten en acciones - Las líneas de flujo indican la secuencia y anidamiento de las declaraciones

Para el ejemplo del paraguas, el pseudocódigo podría verse así:

```
INICIO
    SI actualmente_lloviendo ENTONCES
        Llevar paraguas
        Salir afuera
    SI NO
        Revisar pronóstico
        SI se_pronostica_lluvia ENTONCES
            Llevar paraguas
            Salir afuera
        SI NO
```

```
            Dejar paraguas en casa
            Salir afuera
        FIN SI
    FIN SI
FIN
```

Paso 3: Compara representaciones

Reflexiona sobre las diferencias entre las representaciones de diagrama de flujo y pseudocódigo: - ¿Cuál es más fácil de crear? - ¿Cuál es más fácil de entender a simple vista? - ¿Cuál proporciona más detalle? - ¿Cuándo podría ser más útil cada representación?

Parte 5: Del pseudocódigo al lenguaje natural

Ahora practiquemos el proceso inverso: convertir pseudocódigo a lenguaje natural. Esto ayuda a asegurar que tu pseudocódigo sea correcto y completo.

Paso 1: Estudia el ejemplo

Aquí hay pseudocódigo para encontrar el promedio de una lista de números:

```
INICIO
    ESTABLECER suma = 0
    ESTABLECER contador = 0
    MIENTRAS haya más números por procesar
        OBTENER siguiente_numero
        ESTABLECER suma = suma + siguiente_numero
        ESTABLECER contador = contador + 1
    FIN MIENTRAS
    SI contador > 0 ENTONCES
        ESTABLECER promedio = suma / contador
        MOSTRAR promedio
    SI NO
        MOSTRAR "No hay números para promediar"
    FIN SI
FIN
```

Traducción a lenguaje natural: “Para encontrar el promedio de una lista de números, comienza estableciendo la suma y el contador en cero. Mientras haya más números por procesar, obtén el siguiente número, agrégalo a la suma y aumenta el contador en uno. Después de procesar todos los números, si el contador es mayor que cero, calcula el promedio dividiendo la suma por el contador y muestra el resultado. De lo contrario, muestra un mensaje de que no hay números para promediar.”

Paso 2: Traduce al lenguaje natural

Convierte cada uno de estos ejemplos de pseudocódigo a lenguaje natural claro:

1. **Verificando la fortaleza de una contraseña**:

```
INICIO
    OBTENER contraseña
    ESTABLECER fortaleza = 0
    SI longitud de contraseña >= 8 ENTONCES
        ESTABLECER fortaleza = fortaleza + 1
    FIN SI
    SI contraseña contiene números ENTONCES
        ESTABLECER fortaleza = fortaleza + 1
    FIN SI
    SI contraseña contiene caracteres especiales ENTONCES
        ESTABLECER fortaleza = fortaleza + 1
    FIN SI
    SI fortaleza == 0 ENTONCES
        MOSTRAR "Contraseña débil"
    SI NO SI fortaleza == 1 ENTONCES
        MOSTRAR "Contraseña moderada"
    SI NO SI fortaleza == 2 ENTONCES
        MOSTRAR "Contraseña fuerte"
    SI NO
        MOSTRAR "Contraseña muy fuerte"
    FIN SI
FIN
```

2. **Haciendo una lista de compras**:

```
INICIO
    ESTABLECER lista_compras = lista vacía
    MOSTRAR "Revisar despensa para artículos que comprar"
    MIENTRAS se necesiten más artículos
        SI el artículo está bajo o vacío ENTONCES
            AÑADIR artículo a lista_compras
        FIN SI
    FIN MIENTRAS
    MOSTRAR "Revisar refrigerador para artículos que comprar"
    MIENTRAS se necesiten más artículos
        SI el artículo está bajo o vacío ENTONCES
            AÑADIR artículo a lista_compras
        FIN SI
    FIN MIENTRAS
    SI lista_compras no está vacía ENTONCES
        MOSTRAR lista_compras
    SI NO
```

```
        MOSTRAR "No se necesitan artículos"
    FIN SI
FIN
```

Paso 3: Evalúa la claridad

Para cada traducción: - Verifica si tu descripción en lenguaje natural captura todos los pasos en el pseudocódigo - Asegúrate de que la lógica y la secuencia permanezcan iguales - Identifica cualquier parte que haya sido difícil de traducir de vuelta al lenguaje natural

Parte 6: La computadora humana

Esta actividad final ayuda a demostrar cómo el pseudocódigo cierra la brecha entre el pensamiento humano y la ejecución de la computadora.

Paso 1: Escribe un algoritmo en pseudocódigo

Crea pseudocódigo para un juego o acertijo simple, como: - Adivinar un número entre 1 y 10 - Jugar piedra-papel-tijeras - Resolver un acertijo simple

Paso 2: Actúa como la computadora

Encuentra un compañero (o imagina uno) que actuará como el "programador" mientras tú actúas como la "computadora". Tu trabajo es seguir las instrucciones del pseudocódigo exactamente como están escritas, sin hacer suposiciones o usar información que no esté explícitamente indicada.

Paso 3: Ejecuta el programa

El "programador" lee cada instrucción en el pseudocódigo, y tú (la "computadora") la ejecutas con precisión. Si hay ambigüedades o errores en el pseudocódigo, debes comportarte como lo haría una computadora—ya sea produciendo un error o haciendo una interpretación específica basada en las reglas del pseudocódigo.

Paso 4: Depura y mejora

Basándote en la ejecución: - Identifica cualquier ambigüedad o error en el pseudocódigo - Revisa el pseudocódigo para hacerlo más preciso y efectivo - Intenta ejecutar la versión mejorada para ver si funciona mejor

Actividades de extensión

1. Patrones de pseudocódigo

Investiga y crea pseudocódigo para estos patrones comunes de programación: - Intercambiar los valores de dos variables - Encontrar el mínimo y máximo en

una lista - Contar ocurrencias de un elemento específico en una lista - Validar entrada de usuario

2. Investigación de algoritmos

Elige un algoritmo famoso (como búsqueda binaria o ordenamiento burbuja), investiga cómo funciona y escribe pseudocódigo para él.

3. Crear una guía de pseudocódigo

Crea una guía de estilo completa de pseudocódigo para tu propio uso, combinando las convenciones de este libro con cualquier estándar adicional que encuentres útil.

Preguntas de reflexión

En tu cuaderno, responde estas preguntas: 1. ¿Cuál fue el aspecto más desafiante de traducir entre lenguaje natural y pseudocódigo? 2. ¿Cómo te ayudó crear pseudocódigo a entender la lógica de diferentes procesos? 3. ¿Cuándo preferirías usar pseudocódigo en lugar de un diagrama de flujo, y viceversa? 4. ¿Cómo podría ayudarte el pseudocódigo en la planificación de tareas complejas en tu vida diaria? 5. ¿Qué ambigüedades en el lenguaje natural se hicieron evidentes cuando intentaste convertirlo a pseudocódigo?

Conexión con la programación

El pseudocódigo es un puente esencial entre el pensamiento humano y la programación de computadoras. Los programadores profesionales a menudo comienzan con pseudocódigo para planificar sus soluciones antes de escribir código real. Las habilidades que has desarrollado en esta actividad se traducirán directamente a la programación en cualquier lenguaje, ya que el pseudocódigo captura la estructura lógica que comparten todos los lenguajes de programación, independientemente de su sintaxis específica.

human computer simulation

Actividad: La computadora humana - Representando programas simples

Visión general

Esta actividad práctica transforma a los estudiantes en "computadoras humanas" que ejecutan código actuando físicamente la lógica y el flujo de programas simples. Al encarnar el papel de un procesador de computadora, los estudiantes obtienen una comprensión más profunda de cómo las computadoras interpretan y ejecutan instrucciones, particularmente la lógica condicional y las estructuras de toma de decisiones.

Objetivos de aprendizaje

- Experimentar de primera mano cómo las computadoras ejecutan instrucciones
- Comprender la naturaleza precisa y literal de la ejecución de programas
- Visualizar el flujo de control en programas con declaraciones condicionales
- Reconocer cómo las computadoras mantienen y actualizan valores de variables
- Desarrollar una intuición para la depuración identificando dónde pueden fallar los programas

Materiales necesarios

- Tarjetas grandes con instrucciones escritas (una instrucción por tarjeta)
- Notas adhesivas o pequeñas libretas para representar variables y sus valores
- Cinta adhesiva para marcar "caminos de ejecución" en el suelo
- Opcional: Objetos relacionados con los escenarios del programa (paraguas, mochila, etc.)
- Opcional: Insignias de roles (ej., "CPU", "Memoria", "Entrada/Salida")

Tiempo requerido

60-90 minutos

Preparación

Antes de la actividad, crea tarjetas de instrucciones para al menos dos programas simples. Cada tarjeta debe contener una instrucción que corresponda a una línea de pseudocódigo. Numera las tarjetas para mostrar la secuencia.

Por ejemplo, para un programa de "Rutina matutina": 1. INICIO 2. SI (está lloviendo) ENTONCES ir a tarjeta #3, SI NO ir a tarjeta #5 3. Tomar paraguas

4. Ir a tarjeta #6 5. No tomar paraguas 6. SI (temperatura < 15°C) ENTONCES ir a tarjeta #7, SI NO ir a tarjeta #9 7. Usar chaqueta pesada 8. Ir a tarjeta #10 9. Usar chaqueta ligera 10. Caminar a la parada de autobús 11. FIN

Parte 1: Introducción a ser una computadora

Paso 1: Explicar la actividad

Explica que en esta actividad, los estudiantes se convertirán en "computadoras humanas", siguiendo instrucciones exactamente como lo haría una computadora. Enfatiza que las computadoras: - Siguen instrucciones paso a paso - No pueden adelantarse o hacer suposiciones - Solo pueden hacer exactamente lo que se les indica - Solo pueden tomar decisiones basadas en condiciones específicas

Paso 2: Asignar roles

Asigna diferentes roles a los estudiantes: - "CPU" - sigue instrucciones y toma decisiones - "Memoria" - mantiene valores de variables (usando notas adhesivas) - "Entrada" - proporciona información del mundo exterior - "Salida" - comunica resultados al mundo exterior

Rota los roles para que todos tengan la oportunidad de experimentar ser la CPU.

Parte 2: Ejecución básica de programa

Programa 1: Rutina matutina

Configuración

- Organiza las tarjetas de instrucciones en orden numérico pero separadas
- Marca caminos en el suelo con cinta para mostrar las diferentes rutas de ejecución
- Establece valores de variables en notas adhesivas (ej., "clima = lluvioso", "temperatura = 10°C")
- Coloca objetos apropiados en diferentes estaciones

Ejecución

1. El estudiante "CPU" comienza en la tarjeta #1 (INICIO)
2. Lee cada instrucción en voz alta y realiza la acción especificada
3. Para declaraciones condicionales, consulta con el estudiante "Memoria" para obtener el valor de las variables
4. Basándose en la condición, sigue el camino apropiado a la siguiente instrucción
5. Si la instrucción actualiza una variable, el estudiante "Memoria" actualiza la nota adhesiva

6. El estudiante “Salida” registra o anuncia cualquier acción de salida
7. Continúa hasta llegar a la tarjeta FIN

Discusión Después de completar el programa: - ¿Cómo fue seguir instrucciones exactamente? - ¿Hubo instrucciones ambiguas? ¿Cómo las resolviste? - ¿Cómo cambió el camino cuando cambiaste los valores de las variables?

Parte 3: Programas más complejos

Programa 2: Comprobando elegibilidad

Crea un programa más complejo que determine si alguien es elegible para una actividad específica basándose en múltiples condiciones:

1. INICIO
2. ESTABLECER elegible = falso
3. OBTENER edad
4. SI edad >= 13 ENTONCES ir a tarjeta #5, SI NO ir a tarjeta #11
5. OBTENER tiene_permiso
6. SI tiene_permiso == verdadero ENTONCES ir a tarjeta #7, SI NO ir a tarjeta #11
7. OBTENER nivel_habilidad
8. SI nivel_habilidad == “principiante” ENTONCES ir a tarjeta #9
9. ESTABLECER elegible = verdadero
10. Ir a tarjeta #11
11. SI elegible == verdadero ENTONCES ir a tarjeta #12, SI NO ir a tarjeta #14
12. MOSTRAR “Puedes unirte a la clase intermedia”
13. Ir a tarjeta #15
14. MOSTRAR “Lo siento, no puedes unirte a esta clase”
15. FIN

Variables a seguir

- edad (ej., 10, 15)
- tiene_permiso (verdadero/falso)
- nivel_habilidad (“principiante”, “intermedio”, “avanzado”)
- elegible (verdadero/falso)

Ejecución Ejecuta este programa varias veces con diferentes valores de entrada para ver cómo cambia el resultado.

Parte 4: Simulación de depuración

Paso 1: Introducir errores

Crea una versión de uno de los programas anteriores con “errores” intencionales como: - Instrucciones faltantes - Verificaciones de condición incorrectas - Bucles

infinitos (caminos que nunca llegan al FIN)

Paso 2: Depurar en grupo

Haz que los estudiantes ejecuten el programa e identifiquen dónde salen mal las cosas.

Paso 3: Corregir los errores

Discutan cómo corregir cada error y modifiquen las tarjetas de instrucciones en consecuencia.

Parte 5: Creando tus propios programas

Paso 1: Diseño en grupo

Divide a los estudiantes en pequeños grupos y haz que cada grupo diseñe un programa simple que: - Use al menos dos variables - Incluya al menos dos puntos de decisión (declaraciones SI) - Tenga un comienzo y final claros - Se relacione con un escenario de la vida real

Paso 2: Crear tarjetas de instrucciones

Cada grupo crea tarjetas de instrucciones para su programa.

Paso 3: Ejecutar los programas de otros

Los grupos intercambian programas y los representan.

Paso 4: Retroalimentación

Los grupos proporcionan retroalimentación sobre los programas de los demás: - ¿El programa era claro de seguir? - ¿Había instrucciones ambiguas? - ¿Había errores o errores lógicos? - ¿Cómo podría mejorarse el programa?

Actividades de extensión

1. Añadir bucles

Introduce estructuras de bucle simples (bucles MIENTRAS o PARA) en tus programas. Marca un camino de “retorno de bucle” en el suelo.

2. Múltiples caminos de ejecución

Ejecuta el mismo programa con diferentes valores de entrada y usa cinta de diferentes colores para marcar cada camino de ejecución, creando un mapa visual de todos los posibles caminos a través del programa.

3. Ejecución concurrente

Simula múltiples “CPUs” ejecutando diferentes partes de un programa simultáneamente, y discute los desafíos de la coordinación.

Preguntas de reflexión

En tu cuaderno, responde estas preguntas: 1. ¿Cómo se sintió ser una “computadora humana”? ¿Qué fue desafiante al respecto? 2. ¿Cómo te ayudó trazar los programas a entender la lógica condicional? 3. ¿Te sorprendió alguno de los caminos de ejecución o resultados? 4. ¿Cómo podría ayudarte esta experiencia cuando escribas tus propios programas en el futuro? 5. ¿De qué manera crees que las computadoras reales difieren de nuestra simulación de “computadora humana”?

Conexión con la programación

El proceso de ejecución paso a paso que experimentaste refleja cómo las computadoras reales procesan el código. Cuando eventualmente programes en una computadora, esta comprensión te ayudará a: - Escribir instrucciones más claras y precisas - Predecir cómo se comportará tu programa con diferentes entradas - Depurar problemas trazando mentalmente la ejecución - Entender cómo la computadora mantiene el estado a través de variables

Esta simulación también demuestra por qué las computadoras necesitan instrucciones tan precisas—solo pueden seguir exactamente lo que se les dice, sin la capacidad humana de inferir, asumir o entender el contexto.

Chapter Summary

Capítulo 2: Resumen

Puntos clave

Lógica básica y toma de decisiones

- La lógica booleana usa solo dos valores: verdadero y falso
- Los tres operadores booleanos básicos son AND (Y), OR (O) y NOT (NO)
- Las tablas de verdad ayudan a visualizar los resultados de operaciones lógicas
- Las decisiones en programación se expresan usando estructuras SI-ENTONCES-SI NO
- La lógica computacional permite modelar decisiones complejas mediante la combinación de operadores

Declaraciones condicionales y diagramas de flujo

- Las declaraciones condicionales permiten que los programas tomen decisiones basadas en condiciones
- Los diagramas de flujo proporcionan representaciones visuales de la lógica de un programa
- Los símbolos estándar en diagramas de flujo incluyen inicio/fin, proceso, decisión y entrada/salida
- Las decisiones complejas pueden representarse usando condiciones compuestas o anidadas
- Los diagramas de flujo son herramientas valiosas para planificar, explicar y depurar lógica

Pseudocódigo

- El pseudocódigo es una forma de expresar algoritmos usando una mezcla de lenguaje natural y estructuras similares a la programación
- Sirve como puente entre el pensamiento humano y los lenguajes de programación formales
- Las convenciones comunes incluyen palabras clave en mayúsculas, indentación y símbolos estándar para operaciones
- El pseudocódigo permite enfocarse en la lógica sin preocuparse por la sintaxis específica del lenguaje
- Traducir entre lenguaje natural, pseudocódigo y diagramas de flujo ayuda a desarrollar precisión en el pensamiento algorítmico

Conceptos introducidos

- **Valores booleanos**: Los dos posibles estados (verdadero/falso) que forman la base de la lógica computacional

- **Operadores lógicos**: AND, OR y NOT, que permiten combinar y manipular valores booleanos
- **Tablas de verdad**: Herramientas para visualizar todas las posibles combinaciones de entradas y salidas para operaciones lógicas
- **Declaraciones condicionales**: Estructuras SI-ENTONCES-SI NO que permiten que un programa siga diferentes caminos basados en condiciones
- **Diagramas de flujo**: Representaciones visuales de procesos y algoritmos usando símbolos estandarizados
- **Pseudocódigo**: Un lenguaje intermedio para expresar algoritmos sin la sintaxis estricta de los lenguajes de programación

Actividades completadas

1. **Tablas de verdad y acertijos lógicos**: Creación de tablas de verdad y resolución de problemas usando lógica booleana
2. **Creando diagramas de flujo para decisiones cotidianas**: Visualización de procesos de toma de decisiones usando símbolos de diagramas de flujo estándar
3. **Traduciendo del lenguaje natural al pseudocódigo**: Conversión de instrucciones en lenguaje natural a pseudocódigo estructurado
4. **La computadora humana**: Representación física de la ejecución de programas simples para entender cómo funcionan las computadoras

Próximos pasos

En el Capítulo 3, exploraremos cómo almacenar y manipular datos, aprendiendo sobre: - Variables y tipos de datos - Operaciones matemáticas y lógicas - Entrada y salida - Funciones y modularidad

Estos conceptos ampliarán nuestra capacidad para crear algoritmos más complejos y útiles, aprovechando las habilidades de pensamiento lógico y estructurado que hemos desarrollado en este capítulo.