

Fundamentos de algoritmos, lógica y lenguajes de programación

Breve descripción:

Este componente formativo está orientado a fortalecer las bases lógico-computacionales que son necesarias comprender y aportar soluciones. Se exploran temas que llevan a entender los fundamentos de algoritmos, estructuras lógicas y paradigmas de programación, con relativo énfasis en la aplicación práctica generada a través de entornos de inteligencia artificial. Su propósito es brindar una comprensión generalizada de los contenidos tratados, con la finalidad de poder entender, analizar, modificar y aplicar los códigos que se puedan ir generando con la inteligencia artificial generativa.

Tabla de contenido

Introducción	4
1. Algoritmo	6
1.1. Características de los algoritmos.....	8
1.2. Tipos de algoritmos	9
1.3. Representación de algoritmos	10
1.4. Estructura básica de un algoritmo	13
1.5. Importancia de los algoritmos	15
1.6. Diferencia entre algoritmo y programa.....	17
1.7. Aplicación de los algoritmos	19
2. Lógica de programación.....	21
2.1. Importancia de la lógica en el desarrollo de software	23
2.2. Elementos fundamentales de la programación.....	25
2.3. Estructuras de control	36
2.4. Funciones	41
3. Lenguaje de programación	46
3.1. Paradigma de programación.....	50
3.2. Lenguajes más utilizados	54
Síntesis	56

Material complementario.....	57
Glosario.....	58
Referencias bibliográficas	60
Créditos.....	61

Introducción

Los algoritmos y la lógica de programación representan los principales fundamentos para entender cómo se resuelven los problemas, de forma estructurada y eficiente, a nivel computacional. Aunque este componente no busca enseñar programación en una forma tradicional y profunda, sí persigue dar a conocer las bases conceptuales, de tal manera, que puedan identificar, interpretar y evaluar el código que las herramientas de inteligencia artificial generativa produzcan. Este conocimiento es clave para que puedan aprovechar al máximo las capacidades que las IA gen ponen a disposición de la humanidad.

La IA gen se encuentra cada vez más presente en el desarrollo de soluciones tecnológicas; es por ello que comprender la lógica que está detrás de los algoritmos y los lenguajes de programación es primordial para lograr una mayor y mejor interacción con estas herramientas. Esto permite que entiendan cómo se descomponen los problemas en pasos lógicos, cómo se toman las decisiones y las repeticiones en los procesos, cómo los lenguajes de programación logran transformar y traducir todas las ideas en instrucciones, lo que va materializándose a través del computador.

En este componente formativo se hace un recorrido por los conceptos de algoritmos, estructuras de control, lógica de programación y lenguajes de programación, con la finalidad de construir una base sólida que facilite la interacción con las IA gen y el análisis y aplicación del código generado por ellas, de tal forma que puedan generar soluciones concretas a desafíos planteados. Para una mejor comprensión, se proporcionarán ejemplos prácticos y representaciones gráficas que faciliten el entendimiento, lo que es ideal para facilitar el entorno de trabajo, en donde

cada vez más, será más estrecha la interacción y colaboración conjunta entre la inteligencia artificial y el ser humano.

1. Algoritmo

Es un conjunto finito de pasos o reglas lógicas y precisas que se deben seguir para dar solución a un problema o realizar una tarea determinada. Los algoritmos se componen de reglas de entrada, de proceso y de salida, que se ejecutan en forma ordenada. Su construcción debe ser clara y exacta, sin dejar espacio a las ambigüedades, tampoco puede ser infinita y se desarrolla para dar una solución útil (Mancilla et al., 2016). Las reglas lógicas pueden ir desde describir los pasos para preparar un almuerzo, cruzar por un semáforo, vestirse para ir al colegio, hasta procedimientos más elaborados como aparecer en los primeros lugares en los buscadores en la web, entre otros ejemplos.

Para poder familiarizarse con los algoritmos, se comparten dos ejemplos desde la cotidianidad del ser humano:

Ejemplo 1. Algoritmo para hacer una empanada costeña

Inicio

Preparar ingredientes: harina, agua, sal, carne, cebolla, aceite, condimentos

Mezclar harina, agua y sal para formar masa

Amasar la mezcla hasta lograr una buena masa

Cocinar carne con cebolla y condimentos para el relleno

Dividir masa en porciones pequeñas y homogéneas

Para cada porción hacer:

Formar un círculo con rodillo

Colocar relleno en el centro

Doblar el círculo para formar media luna

Sellar bordes con los dedos o tenedor

Fritar las empanadas en aceite caliente

Revisar que las empanadas logren su punto dorado

Sacar empanadas y dejar escurrir

Poner las empanadas en la vitrina para la venta

Fin

Ejemplo 2. Algoritmo para asistir al colegio

Despertar a las 5:00 A.M.

Levantarse de la cama

Ir al baño y asearse (lavar cara, cepillar dientes, bañarse)

Ponerse el uniforme del colegio

Desayunar

Cepillarse nuevamente los dientes

Preparar la mochila con los libros y demás útiles escolares

Verificar que todo esté en la mochila (libros, lápices, cuadernos, calculadora)

Despedirse de los padres

Salir de casa hacia el colegio

1.1. Características de los algoritmos

Las principales características con las que tiene que contar un algoritmo para ser de utilidad (Revolledo, 2021) son:

Precisos

Indica el orden de los pasos para llegar correctamente al final, no puede tener espacio para dudas ni ambigüedades.

Ejemplo: En vez de escribir “Haga la suma”, se escribe “Suma $A + B$ ”.

Definidos

Su entrada, proceso y salida tienen que estar bien definidas; un mismo algoritmo debe arrojar el mismo resultado cuando se alimenta con los mismos datos.

Ejemplo: Si el número es 4 y el algoritmo lo multiplica por 2, siempre debe dar 8.

Finitos

El proceso para el cual fue diseñado el algoritmo debe tener un fin, no puede quedarse funcionando en un ciclo infinito. Esto demuestra que cuenta con una serie de pasos determinados para cumplir y arrojar un resultado.

Ejemplo: Contar del 1 al 5 y luego finalizar el proceso. No tiene que seguir contando infinitamente.

Legibles

Lo escrito en el algoritmo debe ser claro y ordenado, para que pueda ser comprendido fácilmente por otras personas.

Ejemplo: No usar frases como “llega a los usuarios con nuevas ofertas”, sino redactar correctamente como “sumar los precios de los productos vendidos”.

Concretos

Cada instrucción o paso lógico debe indicar exactamente qué se debe hacer, sin ambigüedades.

Ejemplo: En vez de escribir el paso “ordenar los objetos”, se debe escribir “ordenar los objetos de menor a mayor utilizando el método de burbuja”.

1.2. Tipos de algoritmos

Según la forma en que se resuelven los problemas, existe una variedad de tipos de algoritmos, los cuales manejan una lógica determinada, que pueden ser utilizados según naturaleza para seguir pasos fijos, tomar decisiones o repetir acciones. Según la necesidad, se puede apoyar en uno de esos tipos para construir una solución más eficiente y adaptada a la situación. A continuación, se describen los diferentes tipos de algoritmos:

Tabla 1. Tipos de algoritmos

IA Gen	Contenido generado	Descripción	Acceso
Secuencial	Es el tipo de algoritmo tradicional, y se encarga de ejecutar una instrucción tras otra, cumpliendo al pie de la letra cada una de las órdenes escritas, en el estricto orden en que están organizadas.	Tareas simples y lineales.	Calcular el área de un triángulo.

IA Gen	Contenido generado	Descripción	Acceso
Condicional o de decisión	Se diseñan para tomar decisiones con base a unas condiciones ya descritas. Por cada condición se realiza una acción.	Validar edades, permisos, opciones.	Determinar si un estudiante del SENA aprueba o no la formación.
Repetitivo o cíclico	Repita las instrucciones, mientras la condición que esté dando esa orden, se esté cumpliendo. Es recomendable cuando las tareas tienen que hacerse varias veces.	Contadores, bucles, procesos automáticos.	Contar del 1 al 5.
Rekursivo	Su nombre se da, porque se llama asimismo para resolver el mismo problema en versiones más pequeñas. Necesita apoyarse en una condición para ser detenida.	Factorial, Fibonacci, problemas divisibles.	Calcular la factorial de un número
Aleatorio o probabilístico	Se apoya en decisiones al azar basadas en probabilidad. A diferencia de otros tipos, aquí una misma entrada no llevará a la misma salida.	Juegos, simulaciones, IA.	Obtener el número de la rifa que se puso a jugar.

1.3. Representación de algoritmos

Los algoritmos pueden ser representados de diferentes formas, según la necesidad o el nivel de conocimiento de la persona. Las formas más conocidas para representar un algoritmo son:

Lenguaje natural estructurado

Es la forma más simple de representar un algoritmo, y su forma consiste en usar frases cortas en un lenguaje cotidiano, con una estructura lógica, que indique en cada línea qué es lo que tiene que ir haciendo el algoritmo. Esta representación es fácil de

entender y estructurar por personas sin conocimiento, pero también puede ser ambigua, si no se redacta con claridad.

Ejemplo 3. Sumar dos números en lenguaje natural

Pedir al usuario el valor del primer número

Pedir al usuario el valor del segundo número

Sumar los dos números

Mostrar el resultado de la suma de los dos números

Pseudocódigo

Es un formato más avanzado que el lenguaje natural estructurado, ya que se utiliza el lenguaje natural, pero en una forma más técnica, llevando la estructura de la codificación de los lenguajes de programación, respetando la lógica computacional. Es de gran ayuda para el análisis de los programadores, sin estresarse o afanarse por la sintaxis de un lenguaje de programación en particular. Se recomienda mucho para el diseño de soluciones antes de programar en un lenguaje; aunque se requiere entender la lógica de programación para aprovechar sus bondades.

Ejemplo 4. Sumar dos números en pseudocódigo

Inicio

Mostrar mensaje: "Ingrese el primer número"

Leer número1

Mostrar mensaje: "Ingrese el segundo número"

Leer número2

Calcular suma como $\text{número1} + \text{número2}$

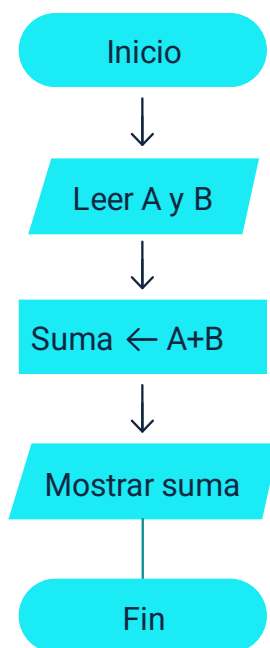
Mostrar mensaje: "El resultado de la suma es:" seguido de suma

Fin

Diagrama de flujo






Es una representación gráfica o visual de un algoritmo, que se apoya en recursos como símbolos ya estandarizados que presentan el flujo de un proceso o los pasos que se siguen para dar solución a un problema. Cada uno de los símbolos o gráficos utilizados representan un tipo de acción (inicio, proceso, decisión, etc.) y las flechas indican el orden que se va siguiendo en la ejecución de los pasos. Ejemplo:

Figura 1. Representación gráfica de un diagrama de flujo



Estos símbolos son esenciales en la elaboración de diagramas de flujo, ya que indican claramente las acciones, decisiones y secuencias que conforman un proceso. A continuación, se describen los símbolos más comunes utilizados en los diagramas de flujo:

Tabla 2. Símbolos más utilizados en el diagrama de flujo

Símbolo	Nombre	Función
	Inicio / Final	Simboliza el inicio y/o el final del algoritmo o de un proceso.
	Línea de flujo	Simboliza el orden de que se va ejecutando el algoritmo. La flecha indica hacia donde se va a seguir o cual es el siguiente paso.
	Entrada / Salida	Simboliza la lectura y/o el ingreso de los datos y la visualización del resultado obtenido de una operación o proceso.
	Proceso	Representa cualquier tipo de operación o acción dentro del algoritmo.
	Decisión	Permite evaluar una condición que puede tener respuesta verdadera o falsa.

1.4. Estructura básica de un algoritmo

Lo invitamos a consultar el siguiente contenido en formato pódcast sobre la estructura básica de un algoritmo, detallando sus componentes fundamentales: entrada, proceso y salida, con ejemplos que facilitan su comprensión.

Podcast. Estructura básica de un algoritmo

¿Te suena la palabra “algoritmo” y no sabes por dónde empezar?

¡No te preocupes! Hoy te explicamos su estructura de forma sencilla.

Un algoritmo es como una receta: una serie de pasos lógicos que nos ayudan a resolver un problema.

Su estructura básica se divide en tres partes: entrada, proceso y salida.

Todo comienza con la entrada, es decir, los datos que el algoritmo necesita para funcionar.

Pueden ser números, textos, listas... ¡lo que sea necesario para empezar!

Imagina que quieres sumar dos números:

la entrada serían esos dos valores iniciales.

Luego sigue el proceso, el conjunto de instrucciones que transforman esos datos.

Aquí se realizan cálculos, se ordenan listas o se toman decisiones.

¡Es el corazón del algoritmo! Todo lo importante ocurre en esta etapa.

Finalmente, llegamos a la salida: el resultado obtenido después del proceso.

Puede ser un número, una lista ordenada o una respuesta lógica.

Así de simple: entrada, proceso y salida.

Tres pasos para entender cómo funciona un algoritmo.

Sigue explorando este y otros temas en nuestros contenidos educativos.

Porque cuando entiendes cómo piensa un algoritmo... ¡empiezas a pensar como un programador!

1.5. Importancia de los algoritmos

La importancia de los algoritmos radica en que constituyen uno de los pilares fundamentales de la informática. Son esenciales para la resolución sistemática y eficiente de problemas complejos, y forman la base sobre la cual se construye la lógica de programación y el desarrollo de software. Su relevancia se manifiesta en múltiples dimensiones, tanto técnicas como prácticas.

Marco lógico para la resolución de problemas. Permiten descomponer cualquier problema en pasos organizados, claros y secuenciales, facilitando su análisis, diseño y solución. Esto simplifica problemas complejos mediante procesos más manejables.

- **Base del desarrollo de software.** Todo programa o sistema es la implementación práctica de uno o varios algoritmos. Un software funcional depende de algoritmos bien definidos que aseguren un comportamiento predecible y correcto.
- **Optimización de recursos computacionales.** Un buen algoritmo mejora el uso del hardware, como el tiempo de ejecución y la memoria, lo que es crucial en contextos de alta demanda, como en la inteligencia artificial o los videojuegos.
- **Reutilización y modularidad.** Los algoritmos pueden ser diseñados para ser reutilizados en distintas partes de un programa o en diferentes proyectos, lo

que promueve la modularidad, las buenas prácticas y facilita el mantenimiento del software.

- **Prevención de errores y ambigüedades.** Un algoritmo bien estructurado asegura el orden lógico de las aplicaciones, evitando errores lógicos y ambigüedades que afectan la confiabilidad del software.
- **Mejora de la comunicación técnica.** Sirven como lenguaje común entre programadores, analistas y diseñadores, favoreciendo la colaboración efectiva dentro de los equipos de desarrollo.

Estas funciones resaltan el valor de los algoritmos como herramientas estructurales clave dentro del desarrollo de software y la ingeniería informática. Sin embargo, su utilidad trasciende lo técnico, alcanzando un impacto más amplio en la vida cotidiana, los negocios y la innovación tecnológica.

- **Adaptabilidad a diferentes lenguajes.** El dominio en la creación de algoritmos permite a los programadores adaptarse fácilmente a distintos lenguajes de programación y estilos de desarrollo.
- **Aplicabilidad en procesos cotidianos.** Más allá del software, los algoritmos son herramientas útiles para describir y organizar procesos personales y empresariales de forma eficiente.
- **Automatización y toma de decisiones.** Son fundamentales en el desarrollo de máquinas autónomas, capaces de ejecutar tareas repetitivas o tomar decisiones basadas en reglas predefinidas, potenciando la automatización en múltiples sectores.

- **Impulso a la innovación tecnológica.** Contribuyen a la creación de productos y servicios más sofisticados, esenciales para ser competitivos y mantener el progreso en contextos empresariales y científicos.
- **Base de la inteligencia artificial.** Los modelos de IA se sustentan en algoritmos avanzados que permiten el procesamiento, análisis y predicción a partir de grandes volúmenes de datos. Sin ellos, sería imposible el desarrollo de sistemas inteligentes que aprendan y se adapten automáticamente.

1.6. Diferencia entre algoritmo y programa

Aunque a menudo se utilizan de manera indistinta, **algoritmo y programa (software)** son conceptos distintos que cumplen funciones complementarias dentro del desarrollo informático. El **algoritmo** es la base lógica y conceptual que describe cómo resolver un problema paso a paso, mientras que el **programa** es la implementación práctica y codificada de uno o varios algoritmos, capaz de ser ejecutado por una computadora. La siguiente tabla compara ambos términos según distintos criterios, destacando sus diferencias clave:

Tabla 3. Diferencia entre algoritmo y software

IA Gen	Algoritmo	Software
Definición	Conjunto de instrucciones organizadas que siguen un orden lógico para resolver un problema o realizar una tarea.	Conjunto de instrucciones codificadas, datos y documentación para resolver un problema o una tarea en un computador.
Naturaleza	Conceptual, abstracto y lógico.	Concreto, físico (código ejecutable) y visible en el computador.

IA Gen	Algoritmo	Software
Representación	Se puede representar el lenguaje natural, pseudocódigo y en diagramas de flujo, etc.	Se representa por medio de código según el lenguaje de programación en que se está trabajando, y puede incluir interfaces, gráficas, archivos, entre otros elementos.
Ejecutabilidad	No puede ejecutarse en la computadora. Para conocer su funcionamiento, pueden utilizarse técnicas como la prueba de escritorio.	Se ejecuta directamente en una computadora o cualquier otro dispositivo informático que soporte un lenguaje de programación.
Dependencia	Su escritura es independiente de cualquier lenguaje de programación, dispositivo electrónico o plataforma tecnológica.	Es dependiente del lenguaje de programación en que sea escrito o de la plataforma tecnológica en que sea ejecutado.
Complejidad	Su escritura es más sencilla, ya que por lo general se limita a la resolución de un problema o tarea en particular.	Es más complejo, ya que sus particularidades conllevan a usar más recursos de los que se podrían utilizar para transcribir literalmente el algoritmo; además pueden incluir múltiples algoritmos, adicionarles otros recursos para robustecer su funcionalidad y relevancia.
Propósito	Describir cómo se puede solucionar un problema o realizar una tarea de forma lógica.	Proporcionar una solución completa que pueda ser implementada para que el usuario final u otro sistema puedan usarla.
Modificación	Puede ser modificada a nivel lógico o a nivel conceptual.	Puede ser modificada a nivel de código, configuración y documentación.

1.7. Aplicación de los algoritmos

Una vez comprendida la naturaleza y estructura de los algoritmos, puede afirmarse que su aplicación es fundamental en una amplia variedad de áreas, tanto en el campo de la informática como en diversos sectores de la vida cotidiana y de la industria. Su capacidad para resolver problemas de manera eficiente y sistemática los convierte en herramientas indispensables para el desarrollo tecnológico, organizacional y social. A continuación, se presentan algunas de las principales áreas donde los algoritmos juegan un papel clave:

- **Desarrollo de aplicaciones y software.** Todo programa parte de una base algorítmica que define su comportamiento y funcionalidad, desde sistemas simples hasta plataformas complejas.
- **Inteligencia artificial y machine learning.** Permiten a las máquinas aprender de los datos, identificar patrones, tomar decisiones y realizar predicciones. Aplicaciones comunes incluyen reconocimiento de voz, visión artificial y sistemas de recomendación.
- **Sistemas de búsqueda y navegación.** Plataformas como Google, sistemas de GPS y redes sociales utilizan algoritmos para proporcionar resultados relevantes, rutas óptimas y conexiones entre usuarios de manera eficiente.
- **Automatización industrial y robótica.** Mediante algoritmos, se controla el funcionamiento de máquinas y procesos automatizados, asegurando precisión, seguridad y eficiencia en entornos industriales.
- **Análisis de datos y Big Data.** Posibilitan el procesamiento de grandes volúmenes de datos, facilitando la identificación de tendencias y la toma de decisiones estratégicas en distintas áreas.

Estas aplicaciones evidencian cómo los algoritmos no solo son fundamentales en el desarrollo tecnológico, sino que también se integran en múltiples contextos donde su uso contribuye a optimizar procesos, incrementar la seguridad y personalizar la experiencia del usuario.

- **Criptografía y seguridad informática.** Garantizan la confidencialidad, integridad y autenticidad de la información mediante técnicas de cifrado y autenticaciones robustas.
- **Sistemas financieros y bancarios.** Se utilizan para controlar transacciones, detectar fraudes, calcular riesgos, gestionar inversiones y fortalecer la seguridad en un sector altamente sensible.
- **Educación y entretenimiento.** Son clave en el diseño de juegos, simuladores y plataformas educativas, que se adaptan al estilo de aprendizaje y preferencias del usuario, ofreciendo experiencias personalizadas e interactivas.

2. Lógica de programación

Muchas personas tienden a confundir la programación con la lógica de programación. Por ello, es importante aclarar que esta última se enfoca en el diseño organizado de soluciones a problemas identificados, sin depender de un lenguaje de programación específico. Esto significa que no se requiere conocimiento previo de un lenguaje en particular para trabajar la lógica de programación. Su aprendizaje y dominio proporcionan una base sólida que facilita la comprensión de cualquier lenguaje y la resolución estructurada de problemas computacionales.

En el capítulo anterior se abordó el concepto de algoritmo. La lógica de programación se conecta directamente con este tema, ya que representa la técnica que permite diseñar algoritmos, es decir, crear sentencias lógicas compuestas por pasos claros, organizados y ordenados para resolver problemas concretos. Esta técnica combina dos principios esenciales: la computación y la lógica matemática (Arias, 2014).

¿Qué permite la lógica de programación?

La lógica de programación, como habilidad, permite estructurar ideas y organizar instrucciones de manera coherente, con el fin de encontrar la solución más eficiente a un problema. En otras palabras, se trata de la **capacidad para identificar un problema, analizarlo y encontrar una solución precisa**, basada en reglas y procesos consistentes que posteriormente se traducen en código. Esta lógica es la base para el desarrollo de todo tipo de programas, desde aplicaciones simples hasta sistemas complejos como navegadores web, calculadoras científicas, sistemas operativos o modelos de inteligencia artificial (Crack The Code, 2023).

Una forma de entenderla: la receta de cocina

Para comprenderla mejor, se puede **comparar con una receta de cocina**: cada paso debe estar bien definido, en orden, sin omitir instrucciones. La computadora, como ejecutora de instrucciones, necesita que cada orden esté escrita con absoluta claridad para evitar confusiones. Así, la lógica de programación organiza un plan paso a paso que guiará a la máquina para que **cumpla con precisión lo que se le ha indicado**.

Aplicación práctica

La lógica de programación permite **solucionar problemas mediante reglas claras y definidas**, y se apoya en ciertos elementos (como estructuras de control, variables, operadores y funciones) que **organizan y conectan esas reglas**. Luego, este conjunto estructurado se convierte en código, el cual será **interpretado por la computadora** para ejecutar y materializar soluciones a los distintos desafíos o necesidades planteadas.

Para mayor claridad, se comparten unas analogías sencillas, que les ayudarán a tener una mayor idea de lo que es la lógica de programación:

- En un rompecabezas, las piezas encajan en un solo lugar y con unas en específico, para que la figura tenga sentido; así es con la lógica, como un reloj cuando engrana sus ruedas dentadas. Cuando se razona sin lógica, las cosas pueden perder el sentido.
- Deseas cocinar un succulento platillo guajiro, para ello sigues la receta al pie de la letra, sin alterar sus ingredientes, ni el orden, respetando sus cortes, adobos y tiempos de cocción. En la lógica de programación es igual, tienes que seguir un proceso ordenado para lograr el resultado deseado.

- Tomas el mapa, ubicas el lugar de salida y de llegada, visualizas las diferentes rutas y tomas el camino más corto y menos congestionado, para llegar a tu destino. La lógica te ayuda a encontrar los mejores caminos, razonablemente hablando, para resolver un problema, tomar una decisión, o desarrollar una idea.

Con todo lo anteriormente explicado, se puede concluir que la lógica de programación, representa la capacidad de estructurar el pensamiento de forma coherente, precisa y sistemática, para trabajar en la construcción de soluciones computacionales sólidas. Es una competencia para la vida, que combina análisis, abstracción, matemáticas y creatividad, lo cual en conjunto brinda destrezas al hombre para trabajar en el desarrollo de software, de manera confiable y eficaz.

2.1. Importancia de la lógica en el desarrollo de software

La lógica de programación es importante, porque representa la base para aprender a programar, fortalecer el razonamiento, generar las habilidades y destrezas para programar con eficacia, brindando recursos para que las personas puedan entender problemas complejos, identificar patrones y buscar soluciones. Incluso, desarrollar esta habilidad, ayuda tanto a niños como a adultos a programar más fácil y más rápido, desarrollando en el camino pensamiento crítico, mejorando capacidades como la resolución de problemas de todo tipo de situaciones que se les vayan presentando en su aprendizaje (Crack The Code, 2023).

La lógica es el corazón de la programación, por eso, más allá de codificar o escribir simples líneas de código, lo que busca es llevar a la persona a pensar de forma clara, eficiente y bien estructurada, para que logre dar soluciones a los problemas. De ahí que dominar la lógica no solo representa mejorar en la habilidad para programar, sino que fortalece el pensamiento crítico, facilita la adaptación a nuevas tecnologías y brinda ayudas para enfrentar diferentes tipos de retos, dentro y fuera del mundo digital (The importance of logic in learning to code, 2024). A continuación se comparten 5 puntos clave sobre la importancia de la lógica en el desarrollo de software:

- **Habilidades para resolver problemas.** Brinda las herramientas para analizar y resolver los problemas paso a paso, anticipando errores y realizando una depuración cuidadosa. Es como armar un rompecabezas: se observa cada pieza, se analiza su forma y se construye la solución poco a poco.
- **Escritura de código fuerte.** Pensar de forma lógica contribuye a redactar un código más claro, ágil, ordenado y fácil de mantener. Es similar a organizar un dormitorio siguiendo un orden práctico que evita retrabajos y mantiene todo bajo control.
- **Comprensión de conceptos de programación.** Tener una base sólida en lógica facilita el aprendizaje de estructuras, algoritmos y paradigmas complejos. Es como contar con una caja de herramientas bien equipada: permite construir desde lo más simple hasta lo más sofisticado.
- **Depuración y solución de errores.** El pensamiento lógico proporciona recursos útiles para identificar fallos y aplicar pruebas que permitan corregirlos. Es

como estar perdido en un laberinto y usar el razonamiento para hallar la salida paso a paso.

2.2. Elementos fundamentales de la programación

Dentro de la lógica de programación, los elementos fundamentales vienen a ser los recursos y las piezas básicas que se necesitan para poder construir cualquier aplicación o sistema. Aquí se habla de los bloques que hacen que el código cobre vida y que permiten que una computadora pueda leerlos, entenderlos y ejecutar las órdenes dadas en las diferentes líneas. Se había hablado de los algoritmos, pero no se habían tratado estos elementos. Ahora, para poder complementar y poder engranar todos los recursos que se necesitan para convertir todas esas ideas en soluciones tecnológicas reales, se van a conocer los principales elementos que intervienen en la programación:

- **Variables.** Son espacios donde se guarda información que puede cambiar.
- **Tipos de datos.** Son las distintas formas de información que puede manejar un programa.
- **Operadores.** Son símbolos que permiten hacer cálculos o comparaciones.
- **Estructuras de control.** Son símbolos que permiten hacer cálculos o comparaciones.
- **Funciones o procedimientos.** Son bloques de código que se pueden reutilizar para realizar tareas específicas.

Variables

Las variables son como contenedores que guardan información o datos, que a su vez se almacenan en la memoria del computador. Cada una de dichas variables es identificada por un nombre, y el valor o dato que este almacena, puede cambiar a lo largo del tiempo, mientras se esté ejecutando el algoritmo o el programa. En otras palabras, pueden ser como las cajas en donde se almacenan cosas en la casa, solo que a cada una de esas cajas se le da un nombre y en vez de almacenar ropa, zapatos o juguetes, se almacenan datos como la edad, el nombre, la ciudad, etc.).

Las variables cumplen con unas características particulares:

- Se identifican con un nombre único, eso quiere decir que no puede haber dos variables con el mismo nombre.
- El valor o dato que se almacena en la variable puede cambiar.
- A la variable, cuyo valor no cambia, se le conoce como constante.
- Las variables, según el lenguaje de programación en donde se estén utilizando, pueden necesitar ser declaradas. Las declaraciones se dan según el tipo de datos que manejará la variable.

A continuación, se presentan dos ejemplos que presentan cómo utilizar variables en un algoritmo:

Ejemplo 5. Asignación de variables

Inicio

```
// Asignar valores directamente
```

```
edad = 18
```

```
nombre = "Carlos"
```

```
// Mostrar los valores
```

```
Mostrar "La edad de", nombre, "es", edad
```

Fin

A continuación, se presentan dos ejemplos que presentan cómo utilizar variables en un algoritmo:

Ejemplo 6. Declaración de variables

Inicio

```
// Declarar variables
```

```
Definir edad como entero
```

```
Definir nombre como texto
```

```
// Asignar valores
```

```
edad = 18
```

```
nombre = "Carlos"
```

```
// Mostrar los valores
```

```
Mostrar "La edad de", nombre, "es", edad
```

Fin

Tipos de datos

Los tipos de datos son los que determinan la clase de información que se almacenará dentro de una variable. Los tipos de datos son fundamentales para programar, ya que según su naturaleza, es que se puede saber qué tipo de operaciones se pueden hacer con las variables, por ejemplo, si la variable es numérica, se podrán hacer sumas o multiplicaciones con ella. Entre los principales tipos de datos se tienen:

Tabla 4. Tipos de datas

IA Gen	Contenido generado	Descripción
Enteros (int)	Número sin decimales	8, 1000, 1200000, -55
Flotantes (float)	Números con decimales	3.14, -0.5, 2.75
Cadenas de texto (string)	Secuencia de caracteres	"Luis", "Finca La Costa", "La Jagua del Pilar"
Booleanos (bool)	Solo dos valores: verdadero o falso	True, false

Ejemplo 7. Declarando tipo de datos

Este ejemplo muestra cómo se declaran variables especificando el tipo de dato que almacenarán, y luego se les asignan valores correspondientes:

Inicio

Definir edad como entero

Definir nombre como texto

edad = 16

nombre = "Juan"

Mostrar "La edad de", nombre, "es", edad

Fin

Operadores

Un operador es un símbolo (+, -, <, %) que se utiliza en la programación para realizar las operaciones sobre los datos, los cuales toman el nombre de operandos (valores, variables, expresiones). De acuerdo a la operación que se quiera realizar, se utiliza el o los operadores más apropiados; por ejemplo, si se desea hacer una suma, se utiliza el operador + para sumar dos o más números.

Para entender mejor la funcionalidad de los operadores, se pueden imaginar cómo las herramientas que se tienen en casa: Se desea cortar un papel, se utilizan las tijeras (operador resta); se desea mirar si una ventana es más ancha que otra, se puede utilizar una regla (operador de comparación) y si se desean pegar dos piezas, se puede usar gota mágica (operador lógico).

Para complementar y entender mejor parte de la jerga que se utiliza en el medio, tener en cuenta esta información:

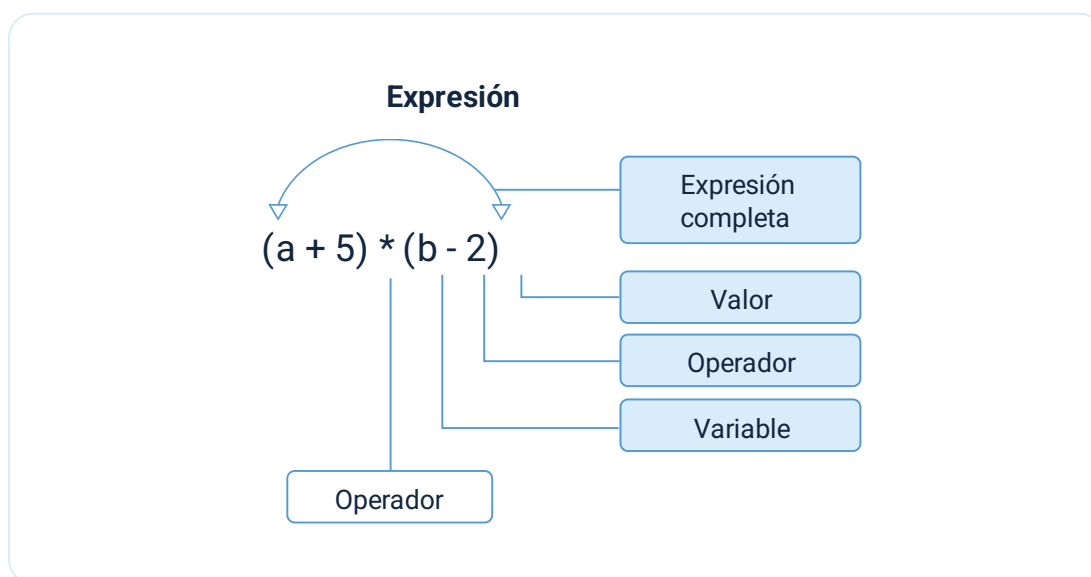
Variable. Espacio de almacenamiento identificado con un nombre, donde se guarda un valor que puede cambiar o no con el tiempo. Ejemplo: en $x = 105$, x es la variable.

- **Valor.** Es el dato que se le asigna a una variable. Ejemplo: en $x = 105$, 105 es el valor asignado a la variable x .
- **Declaración.** Instrucción para que el programa reserve espacio para una variable e indique su tipo. Ejemplo: Definir x como entero declara la variable x como tipo entero.
- **Operandos.** Son variables, constantes o valores involucrados en una operación. Ejemplo: en $x + 105$, x y 105 son los operandos.

- **Expresión.** Combinación de operandos y operadores que produce un resultado. Ejemplo: $(x + 105) * (y - 30)$ es una expresión completa; $(x + 105)$ y $(y - 30)$ son subexpresiones.

En la figura 2 se presenta el ejemplo de una expresión y cómo está conformada, para que la identifiquen mucho mejor.

Figura 2. Vista de una expresión



Operadores aritméticos

Los operadores aritméticos permiten realizar operaciones matemáticas básicas como suma, resta, multiplicación y división. Son fundamentales en todo algoritmo que incluya cálculos numéricos.

Tabla 5. Operadores aritméticos

Operador	Denominación	Descripción	Ejemplo
+	Suma	Suma dos valores	$a + b$
-	Resta	Le resta un valor a otro	$a - b$
*	Multiplicación	Multiplicación dos valores	$a * b$
/	División	Divide un valor entre otro	a / b
%	Módulo	Devuelve el residuo de la división	$a \% b$
^	Potenciación	Eleva un valor a la potencia de otro	$a ^ b$

Operadores relacionales

Los operadores relacionales se utilizan para comparar dos valores. El resultado de estas comparaciones siempre será un valor booleano: verdadero (true) o falso (false). Son esenciales para tomar decisiones en estructuras condicionales.

Tabla 6. Operadores relacionales

Operador	Denominación	Descripción	Ejemplo
==	Igual a	Compara si dos valores son iguales.	$a == b$
!=	Distinto de	Compara si dos valores son distintos.	$a != b$
<	Menor que	Compara si el primer valor es menor que el segundo.	$a < b$
>	Mayor que	Compara si el primer valor es mayor que el segundo.	$a > b$
<=	Menor o igual que	Compara si el primer valor es menor o igual que el segundo.	$a <= b$
>=	Mayor o igual que	Compara si el primer valor es mayor o igual que el segundo.	$a >= b$

Operadores de asignación

Los operadores de asignación se utilizan para almacenar valores dentro de variables. Además del operador de asignación simple (=), existen combinaciones que permiten realizar una operación y asignar el resultado en una sola línea.

Tabla 7. Operadores de asignación

Operador	Denominación	Descripción	Ejemplo
=	Asignación simple	Asigna un valor a una variable.	a = 10
+=	Asignación de suma	Suma un valor a la variable y asigna el resultado.	a += 5
-=	Asignación de resta	Resta un valor a la variable y asigna el resultado.	a -= 3
*=	Asignación de multiplicación	Multiplifica la variable por un valor y asigna el resultado.	a *= 4
/=	Asignación de división	Divide la variable por un valor y asigna el resultado.	a /= 2
%=	Asignación de módulo	Obtiene el residuo de la división de la variable y asigna el resultado.	a %= 3

Operadores lógicos

Los operadores lógicos permiten combinar condiciones y evaluar expresiones booleanas. Son esenciales en estructuras de decisión y bucles, ya que permiten establecer reglas más complejas para ejecutar ciertas instrucciones.

Tabla 8. Operadores lógicos

Operador	Denominación	Descripción	Ejemplo
&&	AND (Y)	Devuelve verdadero si ambos operando son verdaderos.	a && b
	OR (O)	Devuelve verdadero si al menos uno de los operando es verdadero.	a b
!	NOT (NO)	Devuelve el valor contrario del operando (invierte el valor booleano).	!a
XOR	XOR (O exclusivo)	Devuelve verdadero si solo uno de los operando es verdadero, pero no ambos.	a XOR b

En los siguientes ejemplos se comparten algunas aplicaciones de los operadores:

Ejemplo 8. Asignación de variables

Este ejemplo muestra el uso más básico de los operadores de asignación, donde se asignan valores a variables y se muestra la salida. No se realizan operaciones matemáticas, pero sirve como punto de partida para entender cómo se almacenan y utilizan los datos.

Inicio

```
// Asignar valores directamente
```

```
edad = 18
```

```
nombre = "Carlos"
```

```
// Mostrar los valores
```

```
Mostrar "La edad de", nombre, "es", edad
```

Fin

Ejemplo 9. Trabajando expresión por expresión

Aquí se utiliza una estructura paso a paso, ideal para principiantes, que permite observar con claridad cómo se realizan los cálculos intermedios. Se aplican operadores aritméticos para obtener subtotales, calcular impuestos y descuentos, y finalmente el total a pagar. También se hace uso de operadores de asignación para almacenar resultados intermedios.

Inicio

Escribir "Cantidad de guacharaca:"

Leer cg

Escribir "Precio por guacharaca:"

Leer pg

Escribir "Cantidad de acordeón:"

Leer ca

Escribir "Precio por acordeón:"

Leer pa

$subg = cg * pg$

$subp = ca * pa$

$sub = subg + subp$

$iva = sub * 0.19$

$desc = sub * 0.10$

$\text{total} = \text{sub} + \text{iva} - \text{desc}$

Escribir "Subtotal:", sub

Escribir "IVA:", iva

Escribir "Descuento:", desc

Escribir "Total a pagar:", total

Fin

Ejemplo 10. Trabajando todo en una sola expresión

Este ejemplo muestra una versión más condensada del cálculo, utilizando una única línea para calcular el total a pagar. Es útil para comprender cómo combinar varias operaciones aritméticas dentro de una sola expresión, respetando el orden de operaciones mediante el uso de paréntesis.

Inicio

Escribir "Cantidad de guacharaca:"

Leer cg

Escribir "Precio por guacharaca:"

Leer pg

Escribir "Cantidad de acordeón:"

Leer ca

Escribir "Precio por acordeón:"

Leer pa

$$\text{total} = (\text{cg} * \text{pg} + \text{ca} * \text{pa}) + (\text{cg} * \text{pg} + \text{ca} * \text{pa}) * 0.19 - (\text{cg} * \text{pg} + \text{ca} * \text{pa}) * 0.10$$

Escribir "Total a pagar:", total

Fin

2.3. Estructuras de control

Hasta este punto se ha venido trabajando con un flujo secuencial, trabajando dentro del algoritmo línea a línea, en forma vertical, de arriba hacia abajo. En la forma en que se ha venido trabajando, se tiene una sola ruta, y es ideal para problemas sencillos, los cuales necesitan una solución directa, pero cuando ya se habla de tener múltiples caminos, o de poder tomar múltiples rutas, marcadas por las probabilidades que dan la complejidad del problema y la cantidad de datos que intervienen en ella, entonces es recomendable mirar otro tipo de estructuras, para optimizar el algoritmo y atender problemas más complejos.

Cuando el problema es mayor, se necesita tener control de las sentencias que se van a ejecutar y en qué momento se ejecutarán, para ello se echa mano de las estructuras de control.

Las estructuras de control están representadas por recursos o mecanismos que permiten dirigir la ruta o el flujo de la ejecución de un programa. Con las estructuras de control los programas desarrollan la capacidad de tomar decisiones, de poder repetir acciones, de poder ejecutar un sinnúmero de líneas o bloques de código según las condiciones que se tengan planteadas y que se vayan cumpliendo para ir redirigiendo el flujo o simplemente seguir con la ejecución de las instrucciones en forma secuencial.

Estas estructuras permiten que los programas se vuelvan manejables, dinámicos, flexibles y aumenten la capacidad de hacer frente y solucionar problemas con mayor complejidad. Las estructuras de control con las que se trabajan son las condicionales y las iterativas.

Estructura de control condicional

Dentro de las estructuras de control, se tienen las condicionales, las cuales son un tipo de estructura que les permite a los programas tomar decisiones y ejecutar diferentes líneas o bloques de código, mientras que se esté cumpliendo una condición. Dentro de este tipo de estructuras se encuentran:

Condicional simple

Ejecuta una línea de código o un bloque completo, si y solo si se cumple una condición, en otras palabras, si la condición es verdadera.

Ejemplo:

Inicio

Definir edad como entero

edad = 20

Si edad \geq 18 Entonces

Mostrar "Eres mayor de edad"

Fin Si

Fin

Condicional alternativo (Si...Sino)

Ejecuta una línea de código o un bloque de código si la condición es verdadera, y si la condición es falsa, ejecuta una línea o un bloque de código diferente.

Ejemplo:

Inicio

Definir nota como decimal

nota = 2.8

Si nota \geq 3.0 Entonces

Mostrar "Aprobado"

Sino

Mostrar "Reprobado"

Fin Si

Fin

Condicional múltiple (Si...Sino Si...Sino)

Esta estructura condicional analiza y evalúa varias condiciones secuenciales y ejecuta la línea o el bloque de código que cumple con la primera condición deseada, o sea, la primera que devuelve un valor verdadero.

Ejemplo:

Inicio

Definir nota como decimal

nota = 3.5

Si nota ≥ 4.5 Entonces

Mostrar "Excelente"

Sino Si nota ≥ 3.0 Entonces

Mostrar "Aprobado"

Sino

Mostrar "Reprobado"

Fin Si

Fin

Estructura de control iterativa

Las iterativas, también se conocen como bucle o ciclo. Estas le permiten al algoritmo o programa repetir una instrucción o un conjunto de instrucciones varias veces, ya sea porque se está cumpliendo una condición o porque se ha especificado el número de repeticiones.

Muchas veces se da la necesidad de que una tarea necesite realizarse varias veces, como contar números o productos, procesar una lista, o mantener la ejecución de una acción hasta que se desarrolle un evento. De no contar con una estructura como esta, se tendría escribir la instrucción el número de veces que se haría necesario, por ejemplo, al solicitarle al algoritmo que imprima los números del 1 al 100, se tendría que escribir la orden 100 veces, lo cual no es práctico y además es propenso a errores.

Dentro de este tipo de estructuras se encuentran:

Bucle controlado por contador (Para)

Se utiliza cuando se necesita que la acción se repita un número determinado de veces.

Inicio

Para i desde 1 hasta 5 hacer

Mostrar "Número:", i

Fin Para

Fin

Bucle controlado por condición (Mientras)

Se utiliza cuando se necesita que la acción se repita siempre y cuando la condición se esté cumpliendo, o sea cuando la condición sea verdadera.

Inicio

Definir contador como entero

contador = 1

Mientras contador \leq 5 Hacer

Mostrar "Contador es:", contador

contador = contador + 1

Fin Mientras

Fin

2.4. Funciones

Una función está representada por un bloque de código que ha sido diseñado para que ejecute una tarea en particular y que pueda ser reutilizada a lo largo de un algoritmo o de un programa. Dentro de un programa las funciones reciben entradas (argumentos o parámetros), ejecutan el bloque o conjunto de instrucciones, y por lo general, devuelven un resultado, aunque esto último no siempre lo hacen.

No existe un límite que fije la longitud de una función, pero dentro de las buenas prácticas de diseño se recomienda que las funciones cumplan con una sola tarea bien definida; entendiéndose con eso que los algoritmos complejos tendrían que ser divididos en funciones más pequeñas o más sencillas y comprensibles, en la medida de lo posible (Whitney, 2023). La importancia de las funciones dentro de la lógica de programación radica en:

- **Abstracción.** Permite mostrar únicamente lo necesario para usar una función, ocultando los detalles internos y complejos del proceso.
- **Claridad.** Contribuye a que el algoritmo sea más legible, lo cual facilita su comprensión y mantenimiento.
- **Organización.** Permite dividir programas complejos en partes más pequeñas, facilitando su manejo y haciendo el código más flexible.
- **Reutilización.** Las funciones se escriben una sola vez pero pueden usarse muchas veces, lo que evita repeticiones innecesarias, reduce errores, mejora el mantenimiento y optimiza el consumo de recursos del programa.

Las funciones están conformadas por unos componentes que les permiten mantener una estructura para su aprovechamiento dentro del código. Entre sus componentes están:

- **Nombre.** Es la identificación de la función, y para su uso dentro del código, la función es llamada por el nombre, lo que lo hace imprescindible.
- **Parámetros.** Son las variables o los valores que recibe la función para hacer su trabajo y así cumplir con la tarea para la cual fue creada.
- **Cuerpo.** Es el conjunto de instrucciones que hacen el trabajo por el cual fue creada la función.
- **Valor de retorno.** Es el resultado que devuelve la función, después de procesarse, o sea, después de hacer su trabajo. Hay que tener en cuenta que la función no siempre devolverá un valor o un resultado.

A continuación se presentan dos ejemplos comparativos, en donde se puede evidenciar una de las ventajas del uso de funciones:

Ejemplo 11: código redundante vs uso de funciones

Código redundante

Inicio

a = 5

b = 3

c = a + b

Mostrar c

$a = 7$

$b = 2$

$c = a + b$

Mostrar c

Fin

Código con funciones (sin redundancia)

Inicio

Definir función `sumar(a, b)`

Devolver $a + b$

Fin función

$c = \text{sumar}(5, 3)$

Mostrar c

$c = \text{sumar}(7, 2)$

Mostrar c

Fin

Como se puede ver en el ejemplo 12, al utilizar las funciones, se elimina la necesidad de estar repitiendo el mismo bloque de código para sumar dos números. Aquí se está mostrando un algoritmo corto, pero a medida que este vaya creciendo y aumenten la cantidad de líneas de código y se dé la necesidad de estar utilizando la

suma de dos números, no habría que repetir nuevamente el mismo bloque de código, sino llamar la función sumar. Esto hace que el código se mantenga más limpio.

A continuación se presentan en los ejemplos 12 y 13 un comparativo de dos algoritmos, para visualizar como se ve una función cuando retorna un valor en contraste con la que no:

Ejemplo 12. Función con retorno

Inicio

```
// Declarar la función con retorno  
  
Definir función calcularAreaCirculo(radio)  
  
Definir area como decimal  
  
area = 3.14 * radio * radio  
  
Devolver area // Retorna el valor calculado  
  
Fin función
```

Fin

Ejemplo 13. Función sin retorno

Inicio

```
// Declarar una función sin retorno  
  
Definir función mostrarMensaje(mensaje)
```

Mostrar mensaje

// La función muestra el mensaje, pero no devuelve nada

Fin función

// Llamada a la función

mostrarMensaje("¡Hola, estudiantes!")

Fin

Algoritmos

Para una mejor comprensión de los temas vistos hasta el momento, le invitamos a consultar un listado de algoritmos desarrollados, en donde podrán evidenciar los temas tratados en el PDF llamado Anexo_1_Algoritmos que se encuentra en la carpeta de anexos.

3. Lenguaje de programación

Son programas que sirven como herramientas para que los desarrolladores puedan crear otros programas, los cuales organizan los algoritmos y procesos lógicos que se ejecutarán en el computador, controlando el comportamiento y comunicación con el usuario. Se componen de símbolos, de reglas semánticas y sintácticas, con las cuales se construye el código fuente que da vida a otros programas o aplicaciones; además, los lenguajes de programación facilitan el trabajo a los desarrolladores y el equipo de desarrolladores, ya que imitan la lógica de los humanos (Concepto, 2020). Son herramientas que permiten a los programadores darle instrucciones precisas a las computadoras (UNIR, 2024).

Los lenguajes de programación se clasifican en tres tipos de lenguajes (Concepto, 2020):

- **Lenguajes de bajo nivel.** Diseñados para interactuar directamente con hardware específico. No pueden ser utilizados en otros equipos, ya que son exclusivos de un sistema. Su principal ventaja es el aprovechamiento máximo del sistema para el que fueron creados; su mayor desventaja, la falta de compatibilidad con otros sistemas.
- **Lenguajes de alto nivel.** Son universales y compatibles con múltiples sistemas, sin depender de la arquitectura del hardware. Incluyen lenguajes de propósito general y otros con fines específicos.
- **Lenguajes de nivel medio.** Se ubican entre los lenguajes de alto y bajo nivel, aunque no todos los autores reconocen esta categoría. Permiten operaciones de alto nivel y, al mismo tiempo, el manejo de aspectos locales de la arquitectura del sistema.

Los siguientes lenguajes de programación se ubican según su tipo en los siguientes niveles:

Lenguajes de bajo nivel

- Assembler
- Lenguaje máquina
- Ensamblador (x86, ARM)
- AVR Assembly
- PIC Assembly

Lenguajes de alto nivel

- Python
- Java
- JavaScript
- Ruby
- Swift
- Kotlin
- Go

Lenguajes de nivel medio

- C
- C++
- Rust
- Objective-C

Los lenguajes de programación pueden tener otro tipo de clasificaciones según su naturaleza (Volpe, 2021):

Por su compilación:

- Compilado
- Interpretado

Por el paradigma:

- Multiparadigma
- Orientado a objeto (POO)
- Funcional
- Reactivo
- Imperativo

Por el propósito:

- Propósito general
- Propósito específico (DSL)

Por el propósito:

- Propósito general
- Propósito específico (DSL)

Los lenguajes de programación, para funcionar, se apoyan en diferentes componentes (UNIR, 2024), entre los cuales se pueden citar:

Instrucciones

Son los comandos, por medio de los cuales, se les dan las indicaciones a los computadores. Dentro de estas instrucciones pueden estar las operaciones lógicas, aritméticas, las condiciones y los ciclos.

Compiladores e intérpretes

- Lenguajes compilados: son los que se encargan de convertir la totalidad del código fuente que se ha elaborado en un programa a código máquina antes de que este sea ejecutado.
- Lenguajes interpretados: se encargan de leer y ejecutar el código línea a línea. Este procedimiento, aunque ventajoso por un lado, afecta la velocidad y el método en que se ejecutan los programas.

Funciones y estructuras

Le permite a los programadores o desarrolladores definir funciones en las cuales se pueden encapsular bloques de códigos que se pueden seguir reutilizando a lo largo de la escritura del programa. También permiten la organización del código en estructuras de clases, lo que es aprovechable en los lenguajes orientados a objetos. Estos componentes facilitan la modularidad y el mantenimiento que se hace al software, para mantenerlo activo y optimizado.

Los lenguajes de programación también poseen unas características claves que determinan su utilidad y eficacia en el desarrollo del software. A continuación se exponen algunas de ellas (UNIR, 2024):

- **Legibilidad.** El lenguaje debe ser comprensible para cualquier desarrollador, lo que facilita el mantenimiento del software y el trabajo colaborativo.
- **Eficiencia.** Permite realizar tareas rápidamente y optimizar el uso de recursos del sistema.
- **Portabilidad.** El código puede ejecutarse en distintas plataformas sin necesidad de modificaciones importantes.
- **Comunidad y soporte.** Una comunidad activa y con recursos facilita el aprendizaje, la actualización del lenguaje y la resolución de problemas. Lenguajes con baja adopción tienden a desaparecer.
- **Paradigma de programación.** Define el enfoque del lenguaje (funcional, imperativo, orientado a objetos, entre otros), lo cual influye en su adecuación a distintos tipos de proyectos.

3.1. Paradigma de programación

Es el enfoque o el estilo utilizado para desarrollar el software, de tal forma, que estructura y organiza el código según unos principios específicos, para ello hay diferentes formas de diseñar un lenguaje y diferentes formas de trabajar para conseguir objetivos que necesitan los desarrolladores (Canelo, 2020). Son modelos, guías, reglas, métodos, teorías para solucionar problemas a nivel de codificación (Castro, 2021).

Se puede hablar de un paradigma como si fuese un mapa, el cual se utiliza para desplazarse de un punto X a un punto Y; para ello se tienen diferentes rutas, unas más rápidas, otras más lentas, algunas más largas, otras más cortas, pero al final todas permiten llegar al mismo destino. En el mundo de la programación es igual, y los paradigmas representan ese enfoque documentado para escribir el código, donde cada

uno de ellos brinda unas características, ventajas y desventajas según el propósito, pero al final todos los lenguajes logran el objetivo del desarrollo del software (EDteam, 2020).

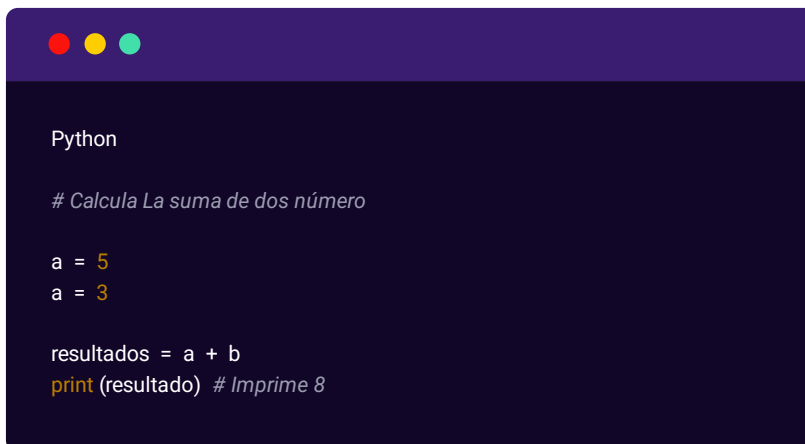
Los principales paradigmas de programación son:

Paradigma imperativo

Basado en instrucciones secuenciales, donde el programador especifica cómo se va a realizar la tarea, paso a paso. Este enfoque detalla un conjunto de instrucciones que transforman el estado de un programa; es decir, los programas escritos bajo este paradigma detallan lo que se debe hacer y cómo se debe hacer. Dentro de este paradigma se encuentran C, Java, Python, JavaScript.

Entre sus características resalta que trabajan con pasos detallados y secuenciales, y el flujo de control lo determinan las estructuras de control como bucles y condicionales. Entre sus principales ventajas, están la flexibilidad, el control sobre el flujo de ejecución, son más fáciles de entender y son recomendables para el desarrollo de tareas sencillas.

Figura 3. Suma de dos números en Python



```
Python

# Calcula La suma de dos número

a = 5
a = 3

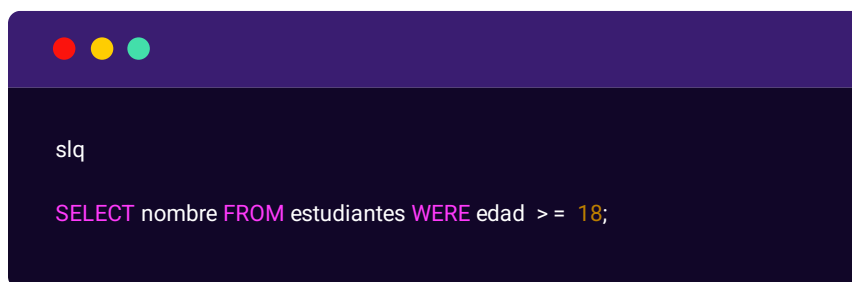
resultados = a + b
print (resultado) # Imprime 8
```

Paradigma declarativo

A diferencia del imperativo en donde se especifica lo que se quiere hacer, aquí se declara como se quiere lograr. Este paradigma está enfocado en el resultado deseado y el sistema es el que se encarga de conseguir la mejor manera de lograrlo. Con este tipo de paradigmas el desarrollador puede escribir programas más concisos y con una menor propensión a los errores. Dentro de este paradigma se encuentran lenguajes como SQL, Prolog.

Entre sus principales ventajas, está el facilitar la escritura con una menor incidencia de errores y su facilidad para que el programador lo entienda mejor, ya que se enfoca más en el resultado.

Figura 4. Consulta de estudiantes mayores de edad en SQL



```
slq  
  
SELECT nombre FROM estudiantes WERE edad >= 18;
```

Paradigma Orientado a Objetos (OOP)

Se centra en objetos y en clases. El objeto es la instancia de una clase, la cual agrupa los datos y comportamientos relacionados. En este paradigma, los problemas son modelados como colecciones de objetos que están interactuando entre sí. Promueve la modularidad, la reutilización de código y la abstracción. Aquí se encuentran lenguajes como Ruby, Java.

Sus 4 pilares son:

- **Encapsulamiento:** agrupa los datos y los métodos en una única entidad.
- **Herencia:** permite la creación de nuevas clases, a partir de clases que ya existen.
- **Polimorfismo:** permite a objetos de diferentes clases ser tratados como objetos de una clase común.
- **Abstracción:** oculta la complejidad de los detalles internos, refiriéndose a cómo es su funcionamiento y por el contrario, muestra solo lo esencial, a quien lo necesita usar. En otras palabras, lo importante no es mostrar cómo se hace, sino lo que hace.

Entre sus principales ventajas se puede decir que mejora tanto la organización como la estructura del código, gracias a la herencia, facilita la reutilización de código. Así mismo, brinda facilidad a la hora de hacer mantenimiento, ya que se puede trabajar con un código más modular.

Figura 5. Definición y uso de una clase en Java



```
public class Persona {
    String nombre;
    int edad;

    public void saludar () {
        System.out.println("hola, mi nombre e " + nombre);
    }
}

public class Mais {
    public static void main (String[] args) {
        persona persona1 = new Persona ();
        persona1.edad = "Juan" ;
        persona1.edad = 25;
        persona1.saludar(); // Imprime: "Hola, mi nombre es Juan"
    }
}
```

3.2. Lenguajes más utilizados

El desarrollo de software representa todo un universo, lleno de diversos lenguajes de programación, cuyas características, formas, estilos, paradigmas los hacen tan diferentes pero a la vez tan ricos, para que cualquier programador encuentre una herramienta que se ajuste a su estilo y necesidades. Eso sí, algunos lenguajes son más populares que otros, tal vez por su facilidad, su versatilidad y, ante todo, por su alta incidencia y adopción en la industria tecnológica.

La idea de dar a conocer un listado de los lenguajes más utilizados en el mundo, permite a los programadores y a los que están comenzando en este mundo de la programación, orientarse a recursos que ofrezcan mayores oportunidades laborales, orientándoles el norte a herramientas que, aparte de facilitarles la creación de soluciones modernas y eficientes, también permite volverse competitivos para aprovechar las oportunidades que brinda el mercado. Claro está, también hay que tener en cuenta que, a mayor popularidad del lenguaje, mayor será su competencia, ya que otros irán por esa estatuilla.

El tener identificado cuáles son los lenguajes más utilizados, no solo ayuda a enfocarse, sino que también les guía para orientarse en una forma eficiente, teniendo de base las tendencias actuales que estén impulsando a las tecnologías con las que hoy en día se está innovando en las industrias, en donde sobresalen áreas como el desarrollo web, las aplicaciones móviles, sistemas empresariales, computación en la nube, internet de las cosas (IoT), realidad aumentada, blockchain, criptomonedas, biotecnología, robótica, computación cuántica y la que viene marcando el hito en los últimos años, la inteligencia artificial.

En esta sección se comparte un listado de los lenguajes de programación más utilizados a nivel mundial, que se han tomado de fuentes actualizadas en los años 2024 y 2025:

Tabla 9. Comparativa de lenguajes más utilizados según diferentes índices (2024–2025)

Posición	Índice TIOBE (mayo 2025)	Índice PYPL (mayo 2025)	Encuesta Stack Overflow (mayo 2024)	GitHub Octoverse (2024)
1	Python	Python	JavaScript	Python
2	C++	Java	HTML/CSS	JavaScript
3	C	JavaScript	Python	TypeScript
4	Java	C/C++	SQL	Java
5	C#	C#	TypeScript	C#
6	JavaScript	R	Bash/Shell	C++
7	Go	PHP	Java	PHP
8	Visual Basic	Rust	C#	Shell (Bash)
9	Delphi/Object Pascal	TypeScript	C++	C
10	SQL	Objective-C	C	Go

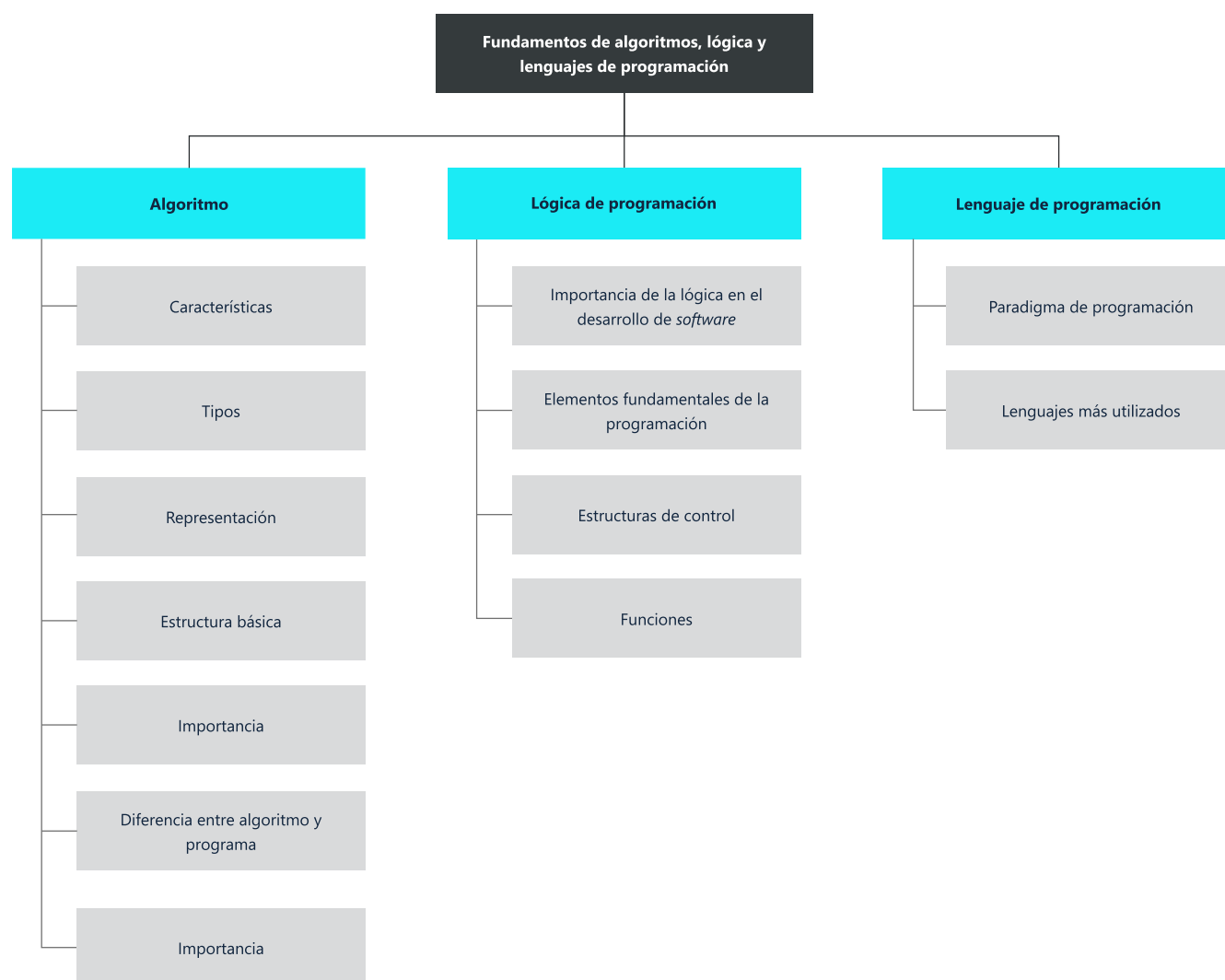
Nota. Elaboración propia con datos de (Jansen, 2025), (PYPL, 2025), (Stack Overflow, 202) y (GitHub, 2024)

Ejemplos

Se comparten unos ejemplos para se pueda conocer la codificación en algunos lenguajes de programación. Lo invitamos a consultar el PDF llamado Anexo_2_Ejemplos que se encuentra en la carpeta de anexos.

Síntesis

A continuación, se presenta una síntesis de la temática estudiada en el componente formativo:



Material complementario

Tema	Referencia	Tipo de material	Enlace del recurso
Algoritmos, estructuras y operaciones	Ecosistema de Recursos Educativos Digitales SENA [@EcosistemaSENAVVirtual]. (2023). Algoritmos, estructuras y operaciones. [Archivo de video] YouTube.	Vídeo	https://www.youtube.com/watch?v=aICQGTU4Dm8
¿Qué es un algoritmo?	Markers, M. [@MagicMarkersPro].(2015). ¿Qué es un algoritmo? [Archivo de video] YouTube.	Vídeo	https://www.youtube.com/watch?v=U3CGMyjzlvM
Pseudocódigos en PSeInt para principiantes	Rayito, M. [@MaestraRayito]. (2021). Pseudocódigos en PSeInt para principiantes. [Archivo de video] YouTube.	Vídeo	https://www.youtube.com/watch?v=70eFuMVEuxg
Como hacer diagramas de flujo con Inteligencia Artificial - ChatGPT	Lagos, F. [@fredislagos22]. (2024). Como hacer diagramas de flujo con Inteligencia Artificial - ChatGPT fredislagos. [Archivo de video] YouTube.	Vídeo	https://www.youtube.com/watch?v=mWcPAGMx8Ic
Lenguajes de programación	Ecosistema de Recursos Educativos Digitales SENA [@EcosistemaSENAVVirtual]. (2022). Lenguajes de programación. [Archivo de video] YouTube.	Vídeo	https://www.youtube.com/watch?v=QpaLtzMslFw
Los 10 lenguajes de programación más usados en 2025.	Maldonado, R. (2025). Los 10 lenguajes de programación más usados en 2025. KeepCoding Bootcamps.	Sitio web	https://keepcoding.io/blog/lenguajes-de-programacion-mas-usados/

Glosario

Algoritmo: es una secuencia de pasos lógicos y finitos que describen cómo resolver un problema o realizar una tarea específica. Cada paso debe ser claro y estar definido de manera precisa. Los algoritmos son la base de cualquier programa de computadora.

Estructuras de control: las estructuras de control son instrucciones que permiten modificar el flujo de ejecución de un programa. Pueden ser condicionales (como if, else) o iterativas (como bucles for, while), y son esenciales para tomar decisiones o repetir tareas.

Funciones: es un bloque de código que realiza una tarea específica, recibe parámetros (opcionalmente), y devuelve un valor (o no). Las funciones permiten modularizar el código, facilitando su reutilización y mantenimiento.

Lenguaje de programación: es un conjunto de reglas y sintaxis que los programadores usan para escribir instrucciones que la computadora pueda entender y ejecutar. Ejemplos incluyen Python, Java, C++ y JavaScript.

Lógica de programación: se refiere al enfoque racional y ordenado que se utiliza para resolver problemas mediante un programa. Implica tomar decisiones, hacer cálculos, manipular datos y controlar el flujo de ejecución del código.

Paradigma de programación: es un enfoque o estilo que define cómo organizar y estructurar el código para resolver problemas. Algunos paradigmas comunes son el imperativo, orientado a objetos, funcional y de programación lógica.

Programa: es un conjunto de instrucciones escritas en un lenguaje de programación que le dice a la computadora cómo realizar una tarea específica. Los programas permiten automatizar procesos, resolver problemas y manipular datos. Estos programas pueden variar desde simples scripts que realizan cálculos, hasta complejas aplicaciones que interactúan con bases de datos o usuarios.

Pseudocódigo: es una forma de describir algoritmos utilizando un lenguaje que se asemeja al lenguaje humano, pero estructurado de manera similar a los lenguajes de programación. No sigue reglas estrictas de sintaxis, pero presenta de forma clara y lógica los pasos necesarios para resolver un problema. El pseudocódigo es útil para planificar y diseñar programas antes de escribir el código real en un lenguaje de programación.

Variables: son espacios de memoria con nombre donde se almacenan valores que pueden cambiar durante la ejecución de un programa. Las variables permiten almacenar datos como números, texto y otros tipos de información que el programa necesita.

Referencias bibliográficas

- Arias, A. (2014). Programación y Lógica Proposicional. IT Campus Academy.
- Crack The Code. (2023). Lógica de Programación: ¿Qué es y Como Mejorarla? Crack The Code. <https://blog.crackthecode.la/logica-de-programacion>
- Mancilla, A., Capacho, J. R., & Ebratt, J. (2016). Diseño y construcción de algoritmos. Universidad del Norte.
- Revollo, A. O. (2021, septiembre 3). Características de un algoritmo, elementos y más: ¡Lo que debes saber para construir el tuyo! <https://www.crehana.com/blog/transformacion-digital/caracteristicas-de-un-algoritmo/>
- The importance of logic in learning to code. (2024). Algocademy.com. <https://algocademy.com/blog/the-importance-of-logic-in-learning-to-code/>
- Whitney, T. (2023). Funciones (C++). Microsoft.com. <https://learn.microsoft.com/es-es/cpp/cpp/functions-cpp?view=msvc-170>

Créditos

Nombre	Cargo	Centro de Formación y Regional
Milady Tatiana Villamil Castellanos	Líder del ecosistema	Dirección General
Olga Constanza Bermúdez Jaimes	Responsable de línea de producción Huila	Dirección General
Armando Javier López Sierra	Experto temático	Centro de Comercio y Servicios – Regional Tolima
Paola Alexandra Moya	Evaluadora instruccional	Centro Agroempresarial y Desarrollo Pecuario - Regional Huila
Yerson Fabian Zarate Saavedra	Diseñador de contenidos digitales	Centro Agroempresarial y Desarrollo Pecuario - Regional Huila
Manuel Felipe Echavarría Orozco	Desarrollador full stack	Centro Agroempresarial y Desarrollo Pecuario - Regional Huila
Carlos Eduardo Garavito Parada	Animador y Productor Multimedia	Centro Agroempresarial y Desarrollo Pecuario - Regional Huila
Andrés Felipe Guevara Ariza	Locución	Centro Agroempresarial y Desarrollo Pecuario - Regional Huila
Aixa Natalia Sendoya Fernández	Validador de recursos educativos digitales	Centro Agroempresarial y Desarrollo Pecuario - Regional Huila
Jaime Hernán Tejada Llano	Validador de recursos educativos digitales	Centro Agroempresarial y Desarrollo Pecuario - Regional Huila
Raúl Mosquera Serrano	Evaluador para contenidos inclusivos y accesibles	Centro Agroempresarial y Desarrollo Pecuario - Regional Huila
Daniel Ricardo Mutis Gómez	Evaluador para contenidos inclusivos y accesibles	Centro Agroempresarial y Desarrollo Pecuario - Regional Huila