

# Fundamentos de programación estructurada y herramientas de desarrollo

## Breve descripción:

Este componente formativo está orientado al reconocimiento, la configuración y el uso eficiente de los editores de código fuente como herramienta fundamental para el desarrollo de software, aplicando la programación estructurada. Se exploran conceptos claves como la estructura de la interfaz de los editores, sus funcionalidades básicas o la configuración del entorno de desarrollo.

## Tabla de contenido

Introducción .....	4
1. Editor de código fuente .....	6
1.1. Importancia .....	8
1.2. Editores populares.....	9
1.3. Interfaz y estructura .....	12
1.4. Instalación .....	15
1.5. Configuración del entorno de trabajo .....	16
1.6. Uso del terminal integrado .....	20
2. Programación estructurada .....	22
2.1. Principios.....	23
2.2. Características.....	24
2.3. Estructuras .....	26
2.4. Funciones y modularidad .....	36
2.5. Ejemplos.....	42
Síntesis .....	68
Material complementario.....	69
Glosario.....	71
Referencias bibliográficas .....	74



Créditos .....	75
----------------	----

## Introducción

La industria del software camina a pasos agigantados, lo que demuestra que su avance acelerado junto al protagonismo que viene teniendo la inteligencia artificial, está llevando a que los profesionales en desarrollo de software sean más demandados; aunque esta actividad ya no es solo exclusividad de ellos, sino que por el contrario, profesionales de otras áreas también se han visto en la necesidad de adquirir conocimientos en este campo, para hacer frente a los desafíos que se van gestando y así mismo, contar con mejores herramientas para realizar y agilizar los procesos en sus áreas profesionales.

Lograr comprender la lógica que hay detrás de un algoritmo, aplicar un diseño estructurado y plasmarlo en un lenguaje de programación, había sido un reto exclusivo de pocos, pero con las herramientas que se tienen hoy en día y los nuevos objetivos que va persiguiendo la sociedad, es mayor el número de personas que ya están trabajando o buscando trabajar en la construcción de software de calidad, que pueda adaptarse fácilmente a los retos del entorno actual y seguirse optimizando, porque así como los desafíos no son estáticos, las soluciones que hoy se están planteando tampoco lo son.

El componente Fundamentos de programación estructurada y herramientas de desarrollo, da respuesta a esa necesidad, brindando un espacio formativo para fortalecer las competencias en la construcción y generación lógica del código fuente, así como del aprovechamiento de los editores de código y demás herramientas de desarrollo que permitan analizar los problemas, plantear una solución lógica o en su defecto interactuar con modelos de inteligencia artificial generativa, para generar

soluciones que se puedan integrar al flujo de trabajo del desarrollo de software. De este modo, este componente contribuye en la adquisición de habilidades que son valoradas en el sector tecnológico, facilitando esa interacción entre los desarrolladores, las herramientas de IA y el entorno profesional.

## 1. Editor de código fuente

El editor de código fuente es una herramienta utilizada para escribir, corregir, modificar y gestionar código en distintos lenguajes de programación. Su uso resulta fundamental en el desarrollo de software, ya que permite crear aplicaciones y programas para entornos de escritorio, móviles o web. Además, facilita la depuración de errores, promueve el trabajo colaborativo —especialmente al integrarse con sistemas de control de versiones— y contribuye a una mayor organización en el proceso de desarrollo (Id Digital School, 2024).

Es un programa diseñado para redactar y editar código, ofreciendo funciones que optimizan el trabajo del programador, tales como autocompletado, resaltado de sintaxis e indentación automática. Suele consumir pocos recursos del sistema, y su disponibilidad varía según el sistema operativo y el modelo de licenciamiento (gratuito o de pago). La elección del editor depende en gran medida de las preferencias personales del desarrollador (UNIR, 2023).

Se trata, en esencia, de una herramienta especializada en la edición y mantenimiento del código fuente en uno o varios lenguajes de programación. A diferencia de los editores de texto convencionales, los editores de código están diseñados para optimizar el trabajo del programador mediante características como navegación por la estructura del proyecto, autocompletado inteligente, detección temprana de errores, entre otras funcionalidades.

Pueden funcionar de manera independiente o integrarse dentro de un entorno de desarrollo integrado (IDE). Su correcta elección influye directamente en la productividad y en la calidad del producto final. Entre los más populares se encuentran Visual Studio Code, Sublime Text, Vim e IntelliJ IDEA. En el flujo de trabajo

actual, el editor de código fuente se ha convertido en una herramienta indispensable, permitiendo centralizar el diseño, la construcción y la evolución del software con mayor organización y control.

Dentro de las principales ventajas por las cuales se les recomienda a los programadores utilizar un buen editor de código fuente están (Id digital school, 2024):

- **Resaltado de sintaxis.** Facilita la identificación de la estructura del código y la detección de errores sintácticos mediante el uso de colores que diferencian palabras clave, funciones, variables, entre otros.
- **Autocompletado.** Ayuda en la escritura del código al sugerir automáticamente palabras clave, variables, funciones y estructuras comunes, según el lenguaje de programación utilizado.
- **Depuración integrada.** Permite identificar y corregir errores directamente desde el editor, con funciones como puntos de interrupción, ejecución paso a paso y seguimiento de variables, sin necesidad de herramientas externas.
- **Control de versiones.** Integra herramientas como Git para gestionar cambios en el código, hacer seguimiento del historial y facilitar la colaboración entre desarrolladores.
- **Extensibilidad.** Admite la instalación de extensiones o plugins para personalizar y ampliar las funcionalidades del editor, adaptándolo a diferentes lenguajes, frameworks o flujos de trabajo.

## 1.1. Importancia

Los editores de código fuente son importantes, ya que en ellos se centraliza el trabajo del desarrollo de software, convirtiéndose en la herramienta principal, en donde los programadores escriben, organizan, administran y mantienen el código que da vida a los programas, sistemas y demás plataformas digitales. Se puede decir que relevancia estaba basada en aspectos claves cómo:

- **Eficiencia en la programación.** Funcionalidades como la navegación rápida por archivos, autocompletado, atajos de teclado y formateo automático aceleran la escritura del código y reducen errores.
- **Prevención y detección de errores.** El resaltado de sintaxis permite identificar errores comunes como palabras mal escritas, paréntesis sin cerrar o estructuras mal formadas durante la escritura.
- **Organización del código.** Permite estructurar archivos y carpetas, aplicar indentación automática y trabajar con múltiples lenguajes, lo que facilita el mantenimiento del software.
- **Personalización y extensibilidad.** Los editores modernos admiten extensiones, configuraciones y temas que adaptan el entorno a las necesidades del programador.
- **Integración con otras herramientas.** Se pueden integrar con sistemas de control de versiones, depuradores, terminales, plataformas en la nube y gestores de paquetes, centralizando el flujo de trabajo.
- **Compatibilidad y portabilidad.** La mayoría de los editores son multiplataforma, lo que permite trabajar en diferentes sistemas operativos sin pérdida de configuración o rendimiento.



## 1.2. Editores populares

A la hora de elegir un buen editor de código para trabajar hay que tener en cuenta el tipo de proyecto en que se está trabajando o se desea desarrollar, los objetivos de programación que se están persiguiendo y el nivel de habilidad que se ha logrado como programador, con el fin de trabajar con el editor que más se ajuste a las necesidades. A la hora de elegir un editor, se recomienda tener en cuenta puntos como rendimiento optimizado, características del editor, funciones de navegación, código de referencias y que tan personalizable es (Weisheim, 2022). A continuación se comparte una lista de editores, donde podrán encontrar un alista de herramientas para facilitar el trabajo de la programación:

**Tabla 1.** Lista de editores de código fuente

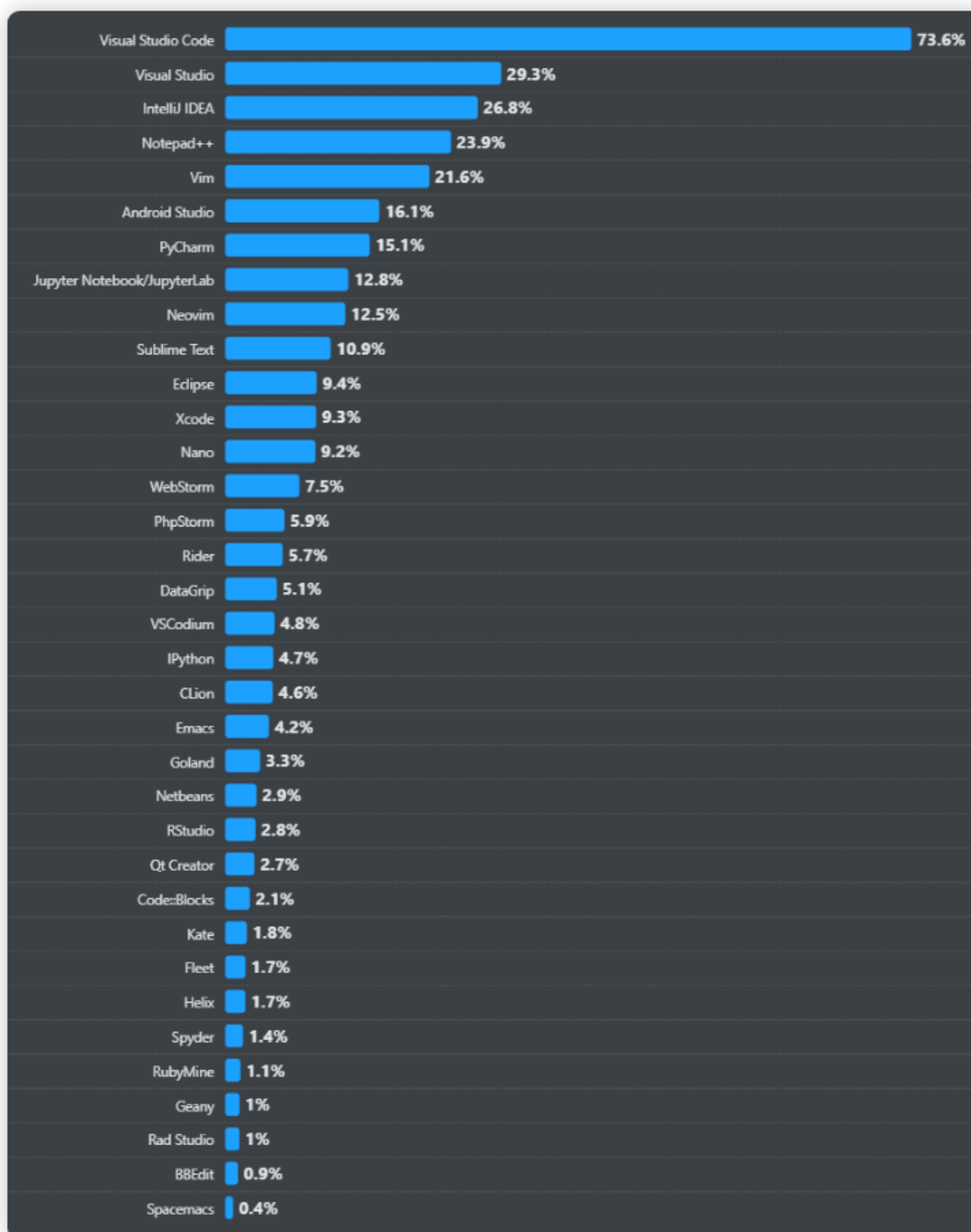
Nombre	Licencia	Lenguajes compatible	Plataformas compatibles	Link de descarga
Visual Studio Code	Gratuita	JavaScript, TypeScript, Python, C++, entre otros	Windows, Linux, macOS	<a href="#">Ir al sitio web</a>
Sublime Text	Gratuita (99 USD por licencia)	C++, Python, PHP, Rails y más	Windows, Linux, macOS	<a href="#">Ir al sitio web</a>
Atom	Gratuita (descontinuado)	C, C++, COBOL, HTML, CSS, Java, PHP, Ruby, Scala, SQL	Windows, macOS, Linux	<a href="#">Ir al sitio web</a>
Notepad++	Gratuita	Más de 70 lenguajes como HTML, CSS, JavaScript, PHP, XML, C++, Swift	Windows, Linux, UNIX	<a href="#">Ir al sitio web</a>
Vim	Gratuita	Casi todos los lenguajes	Unix, Windows, macOS y más	<a href="#">Ir al sitio web</a>
NetBeans	Gratuita	HTML5, C++, JavaScript, PHP, Java y más	Windows, Linux, macOS, BSD	<a href="#">Ir al sitio web</a>

Nombre	Licencia	Lenguajes compatible	Plataformas compatibles	Link de descarga
TextMate	Gratuita	AppleScript, C++, HTML, CSS, Java, PHP, Python, SQL y más	macOS	<a href="#">Ir al sitio web</a>
GNU Emacs	Gratuita	C, C++, Python, Lisp y más	Windows, Linux, macOS, BSD	<a href="#">Ir al sitio web</a>
Codeshare.io	Gratuita	PHP, C#, SCSS, CSS, HTML	Navegadores web	<a href="#">Ir al sitio web</a>

Nota. Elaboración propia con datos de (Weisheim, 2022), (UNIR, 2023)

Basados en la tabla anterior, donde se muestra una variedad de editores, elegir un buen entorno de programación es primordial para la productividad y la experiencia del desarrollador. De ahí que cuando se está comenzando, es bueno conocer sobre los diferentes rankings que se realizan a nivel mundial para ver cómo están posicionadas las herramientas que se utilizan en el campo de la informática, con el fin de estar al día con las tecnologías que se trabajan en el mercado. En ese orden de ideas, la encuesta global de Stack Overflow, recopila una gran cantidad de datos, basado en el consenso de miles de profesionales para identificar las más utilizadas en diferentes áreas de la industria. Los resultados del 2024 posicionan al editor Visual Studio Code como líder indiscutible, con una preferencia del 73,6 %, lo que demuestra actualmente su hegemonía.

**Figura 1.** Ranking de editores de código del año 2024

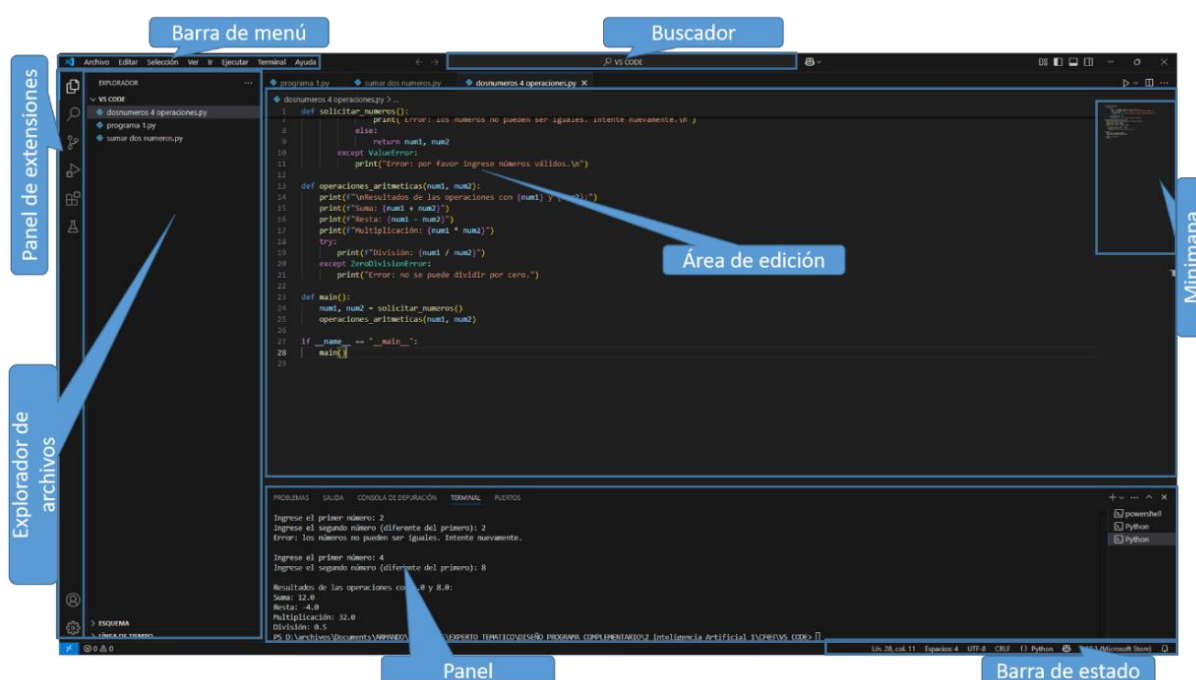


Nora. Tomado de Stack Overflow (2024).

### 1.3. Interfaz y estructura

Hasta este punto, está bien definido que los editores de código fuente han sido diseñados para facilitar la escritura, lectura y gestión del código. Esto se debe, en parte, a la interfaz y a la estructura del editor, que permiten al programador interactuar de forma más fluida con la herramienta. Es importante tener presente que, aunque existe una amplia variedad de editores, cada uno posee particularidades propias. No obstante, la mayoría comparte ciertos componentes comunes que les otorgan una organización tanto visual como funcional. Entre estos componentes comunes, se destacan los siguientes:

**Figura 2.** Interfaz de Visual Studio Code con sus componentes principales.



## **Barra de menú**

Ubicada en la parte superior. Contiene funciones como:

- Archivo: abrir, guardar, cerrar archivos.
- Editar: copiar, pegar, buscar, reemplazar.
- Ver: mostrar u ocultar paneles.
- Ejecutar / Depurar: iniciar scripts, abrir consola o debugger.
- Extensiones / Preferencias: configuración del editor.

## **Explorador de archivos (panel lateral)**

Permite visualizar y navegar por archivos y carpetas del proyecto.

- Crear, modificar o eliminar archivos y carpetas.
- Abrir archivos en diferentes pestañas.
- Ver la estructura general del proyecto.

## **Área de edición (editor de texto)**

Espacio principal para escribir el código fuente. Incluye:

- Resaltado de sintaxis.
- Numeración de líneas.
- Plegado de bloques.
- Autocompletado y sugerencias.

## **Panel de extensiones o plugins**

Permite buscar, instalar y gestionar extensiones para ampliar las funciones del editor:

- Soporte para nuevos lenguajes.
- Integración con bases de datos.
- Sistemas de control de versiones.
- Herramientas de depuración e inteligencia artificial.

## **Consola**

Terminal integrada para ejecutar comandos y ver resultados o realizar pruebas sin salir del editor.

## **Depurador**

Herramienta para:

- Establecer puntos de interrupción.
- Analizar variables y su comportamiento.
- Ejecutar el código paso a paso.

## **Barra de estado**

Ubicada en la parte inferior. Muestra información contextual como:

- Lenguaje activo.

- Codificación.
- Tabulaciones.
- Mensajes del sistema.
- Estado del proyecto.

## 1.4. Instalación

Antes de iniciar la programación o la generación de código, es fundamental contar con una herramienta que facilite este proceso. Visual Studio Code, conocido también como VS Code, se ha consolidado como el editor de código más utilizado en el ámbito del desarrollo de software, gracias a su ligereza, versatilidad y múltiples funciones integradas. Por esta razón, en esta sección se abordará su instalación. Para ello, se invita a seguir los pasos descritos en el video:

### Video 1. Instalación del editor de código Visual Studio Code



[Enlace de reproducción del video](#)

### **Síntesis del video: instalación del editor de código Visual Studio Code**

En este video se explica detalladamente cómo realizar la instalación del editor de código Visual Studio Code en un equipo con sistema operativo Windows. El proceso incluye la descarga desde el sitio oficial, la ejecución del instalador, la aceptación de los términos de licencia, y la configuración inicial con las opciones predeterminadas.

Al finalizar la instalación, se realiza una verificación para comprobar que el programa se ha instalado correctamente. Además, se brinda una introducción sobre cómo configurar el editor según el lenguaje de programación que se desee utilizar, destacando la posibilidad de instalar extensiones que adaptan el entorno de trabajo a diferentes tecnologías.

Este contenido está dirigido a quienes están comenzando en el desarrollo de software y buscan una herramienta confiable y personalizable para programar de forma eficiente.

## **1.5. Configuración del entorno de trabajo**

Después de instalado el editor Visual Studio Code, el siguiente paso es realizar los ajustes al entorno de trabajo según las necesidades del desarrollador. Para ello se realizará una configuración básica inicial del editor y del lenguaje de programación de preferencia, el cual para esta ocasión se hará con Python, por ser uno de los lenguajes de programación más utilizados en el mundo y el de mayor crecimiento por el aporte que viene brindando a la inteligencia artificial y a otras tecnologías disruptivas.



Dentro de la configuración del editor es posible elegir entre temas claros u oscuros para ajustar el color del entorno, cambiar el idioma a uno de preferencia, activar el guardado automático e instalar las extensiones necesarias, por ejemplo, para trabajar con un lenguaje específico. Por ello, se invita a revisar el video donde se explican los pasos para configurar el editor:

## Video 2. Configuración del editor de código Visual Studio Code



### [Enlace de reproducción del video](#)

#### **Síntesis del video: configuración del editor de código Visual Studio Code**

En este video se muestra cómo personalizar Visual Studio Code después de su instalación, comenzando por cambiar el idioma de la interfaz al español mediante la instalación de la extensión correspondiente.

A continuación, se explica cómo ajustar el tema visual del entorno de trabajo, eligiendo entre opciones claras y oscuras para adaptarlo a las preferencias

personales. También se habilita el guardado automático, una función útil que permite centrarse en la escritura del código sin preocuparse por guardar manualmente los cambios.

Otro aspecto abordado es la activación de sugerencias automáticas en la terminal integrada, lo cual facilita la escritura de comandos al ofrecer autocompletado. Se enseña además cómo abrir una carpeta de trabajo donde se organizarán los archivos del proyecto y cómo crear nuevos archivos dentro del editor.

Por último, se explica cómo activar el minimapa, una herramienta visual que permite desplazarse fácilmente por archivos de código extensos.

Este video está dirigido a quienes están comenzando a utilizar Visual Studio Code y desean conocer las configuraciones básicas que optimizan la experiencia de desarrollo desde el primer momento. También se hace énfasis en la importancia de explorar e instalar extensiones adicionales, según las necesidades y lenguajes de programación que se utilicen.

Para el desarrollo de los ejercicios se utilizará Python como lenguaje de programación de apoyo. Por ello, se invita a revisar el video donde se explican los pasos para su instalación y configuración en el editor.

### Video 3. Instalación de Python y configuración de Visual Studio Code



#### [Enlace de reproducción del video](#)

#### **Síntesis del video: instalación de Python y configuración de Visual Studio Code**

En este video se guía paso a paso el proceso de instalación del lenguaje de programación Python y su configuración dentro del entorno de desarrollo Visual Studio Code.

Se comienza con la descarga del instalador oficial desde el sitio web de Python, incluyendo la habilitación de la opción para agregar Python al sistema (Add Python to PATH) antes de proceder con la instalación.

Una vez finalizada la instalación de Python, se abre Visual Studio Code para realizar la configuración necesaria. Se accede al panel de extensiones y se instalan las

herramientas recomendadas para trabajar de manera efectiva con Python. Entre ellas se encuentran:

- **Python (Microsoft):** extensión principal que permite ejecutar y depurar código Python.
- **Pylance:** complemento que proporciona análisis estático avanzado y autocompletado eficiente.
- **Python Debugger (debugpy):** instalado automáticamente, permite depurar código en tiempo real.
- **Pylint:** herramienta que ayuda a mantener un código limpio, legible y alineado con buenas prácticas, ideal para proyectos colaborativos.

El video también resalta la importancia de seguir explorando nuevas extensiones a medida que se adquiere experiencia, con el fin de adaptar el entorno de trabajo a necesidades más complejas o específicas.

Este contenido está dirigido a estudiantes, desarrolladores principiantes o cualquier persona interesada en empezar a programar en Python desde Visual Studio Code, con una configuración adecuada desde el inicio.

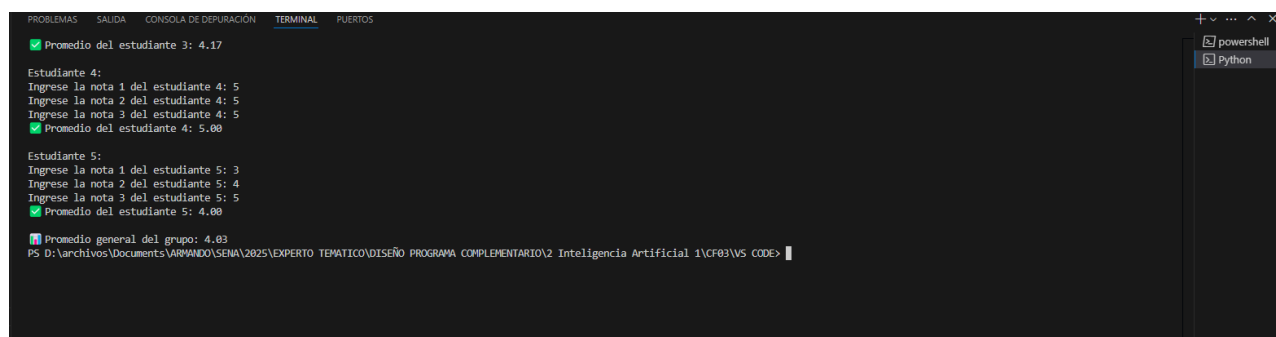
## 1.6. Uso del terminal integrado

Muchos de los editores de código incluyen un terminal integrado para que se puedan ejecutar directamente en ellos comandos desde el editor, sin la necesidad de abrir o utilizar una terminal externa. Esto permite acelerar el flujo de trabajo y mejorar la productividad cuando se trabaja a nivel profesional.

En el caso del editor Visual Studio Code, contiene su propio terminal, lo cual lo hace una herramienta más robusta y autosuficiente para el desarrollo de programas y demás aplicaciones. Esta funcionalidad elimina la incomodidad que representa en el programador el estar cambiando entre una ventana y otra, al tener que estar utilizando herramientas externas al momento de necesitar ejecutar comandos, tener que compilar programas, iniciar servidores locales, ejecutar scripts, instalar dependencias o trabajar con sistemas de control de versiones como Git.

En el caso de Visual Studio Code, el terminal integrado actúa como una línea de comandos que se encuentra alojada en el editor y que se adapta al sistema operativo con el cual se esté trabajando, lo que agiliza el trabajo al permitir esa interacción fluida entre el código y la consola sin salir del entorno de desarrollo. Este editor permite trabajar con múltiples terminales en paralelo, así mismo elegir diferentes intérpretes de comando como CMD, Bash, o PowerShell, entre otros.

**Figura 3.** Visualización de terminal integrada en Visual Studio Code.



```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS
✓ Promedio del estudiante 3: 4.17

Estudiante 4:
Ingrese la nota 1 del estudiante 4: 5
Ingrese la nota 2 del estudiante 4: 5
Ingrese la nota 3 del estudiante 4: 5
✓ Promedio del estudiante 4: 5.00

Estudiante 5:
Ingrese la nota 1 del estudiante 5: 3
Ingrese la nota 2 del estudiante 5: 4
Ingrese la nota 3 del estudiante 5: 5
✓ Promedio del estudiante 5: 4.00

Promedio general del grupo: 4.03
PS D:\archivos\Documents\ARMANDO\SENA\2025\EXPERTO TEMATICO\DISEÑO PROGRAMA COMPLEMENTARIO\2 Inteligencia Artificial 1\CF03\VS CODE> |
```

## 2. Programación estructurada

La programación estructurada es una de las formas de utilizar más utilizadas, sobre todo cuando se inicia en el mundo académico de la programación, y pertenece al paradigma de programación imperativo, cuya finalidad está basada en ir indicando paso a paso cómo resolver un problema. Su enfoque radica en buscar que el código sea sencillo, claro, ordenado y de fácil mantenibilidad, utilizando subrutinas y las estructuras de control (secuencia, selección e iteración):

- **Subrutina.** Bloque de código que se puede reutilizar para que desarrolle una tarea específica y pueda ser llamada en un programa desde diferentes partes del mismo, lo cual es de ayuda para mejorar la organización y la modularidad.
- **Secuencia.** Instrucciones que cumplen con un estricto orden para ejecutarse.
- **Selección.** Instrucciones para toma de decisiones en el flujo de un programa, basado en la ejecución de diferentes bloques de códigos según las decisiones seleccionadas.
- **Iteración.** Instrucciones que se repiten mientras se esté cumpliendo una condición, realizando con ello procesos repetitivos.

Esta forma de programar surge del teorema del programa estructurado de Böhm y Jacopini en el año 1966, y fue apoyada por Dijkstra en 1968, quien promovió la eliminación del uso de instrucciones como goto, porque se tornaban confusas haciendo tediosa la programación; logrando que hoy en día se pueda contar con lenguajes de programación como Java, C, Python y C++, los cuales soportan la programación estructurada (UNIR, 2022).

## 2.1. Principios

Antes de comenzar con la escritura de código, es bueno conocer algunos fundamentos que ayudaran a codificar de manera clara, confiable y organizada; para ello es bueno conocer unos principios básicos en la programación estructurada, que servirán de guía para trabajar en el desarrollo de software con la mejor calidad y adoptando buenas prácticas:

- **Uso de estructuras de control básicas.** Se deben emplear tres estructuras clave: secuencia, selección (if, switch) e iteración (for, while, do-while) para estructurar adecuadamente un programa.
- **Modularización del código.** El programa debe dividirse en subrutinas, funciones o procedimientos que realicen tareas específicas, facilitando la organización, reutilización y mantenimiento del código.
- **Claridad y legibilidad.** El código debe ser comprensible para cualquier desarrollador. Se recomienda usar nombres descriptivos, comentarios explicativos y una organización ordenada del código.
- **Evitar instrucciones de salto incontrolado.** No se deben usar sentencias como goto, ya que dificultan el seguimiento del flujo y la comprensión del código.
- **Diseño descendente.** El desarrollo debe iniciar con un análisis general del problema, descomponiéndolo en subproblemas que se resuelven mediante módulos, funciones o subrutinas.
- **Prueba y verificación sistemática.** El código debe permitir una verificación formal y pruebas que confirmen que cumple sus objetivos y que puede corregirse fácilmente si se requieren ajustes.

## 2.2. Características

A continuación, se presenta un pódcast que resume las cualidades esenciales de un programa bien estructurado, con énfasis en claridad, modularidad y facilidad de mantenimiento.

### Podcast. Fundamentos de la innovación estratégica

**Andrés (con curiosidad):**

¡Hola, hola! Bienvenidos a este nuevo episodio. Yo soy Andrés...

**Carolina:**

Y yo, Carolina. Hoy vamos a hablar de algo clave para quienes programamos:

¿Cómo saber si un programa está bien diseñado?

**Andrés:**

Porque escribir código no es solo que funcione... también importa cómo está organizado.

Y eso, créanme, hace toda la diferencia.

**Carolina:**

Total. Un programa bien diseñado tiene una estructura clara, lógica y fácil de entender.

Por ejemplo, debe estar dividido en módulos reutilizables, así no tienes que repetir código cada vez.



**Andrés:**

También debe ser legible, para que cualquier otro desarrollador pueda leerlo sin adivinar lo que quisiste hacer.

Ya sabes: nombres claros, comentarios útiles y nada de enredos innecesarios.

**Carolina:**

Además, los problemas grandes se deben dividir en partes más pequeñas, para analizarlos mejor y detectar errores con facilidad.

¡Eso se llama diseño modular!

**Andrés:**

Y no olvidemos la importancia de probar por bloques.

Eso permite verificar que cada parte funcione antes de integrar todo.

**Carolina:**

Por último: el código debe ser flexible y escalable, o sea, que pueda adaptarse a nuevos requerimientos o tecnologías sin reescribirlo todo.

**Andrés:**

Así que ya lo sabes: si tu código es modular, legible, flexible y fácil de probar...

¡vas por buen camino!

**Carolina:**

Gracias por escucharnos. ¡Nos vemos en el próximo episodio, donde seguimos programando ideas con inteligencia!

## **2.3. Estructuras**

Todo programa escrito bajo el estándar de la programación estructurada sigue una lógica de programación que permite resolver problemas de una forma sistemática y organizada por medio de tres estructuras básicas: secuencia, selección e iteración. El dominar estas tres estructuras les permite a los programadores tener el control del flujo de un programa, tomar decisiones y automatizar los procesos repetitivos; con lo cual llegan a lograr soluciones eficientes. A continuación se ahonda en cada una de estas tres estructuras, con el fin de que se puedan conocer, dominar y aplicar en futuros desarrollos de software:

### **Estructura de secuencia**

Representa la estructura de control más básica en la programación estructurada, y su principal función consiste en la ejecución de las instrucciones de un programa paso a paso, línea a línea, en el orden en que se han escrito en el código, en orden descendente, desde el inicio hasta el final del código.

Mientras no se utilice una estructura de selección o iteración, cada instrucción se ejecuta de una vez, garantizando con ello un flujo lineal de ejecución, que puede llevar un seguimiento simple y predecible, al que incluso se le pueden realizar fácilmente pruebas para verificar la eficacia del código.

Esta estructura representa la base del código, ya que las estructuras de selección e iteración están escritas bajo la base de la estructura de secuencia.

Para conocer como se ve la estructura de secuencia, se comparten unos ejemplos:

#### **Código de Python**

```
edad_actual = 25

edad_futura = edad_actual + 5

print("Edad dentro de 5 años:", edad_futura)
```

#### **Código en C++**

```
#include <iostream>

using namespace std;

int main() {

    int edad_actual = 25;

    int edad_futura = edad_actual + 5;

    cout << "Edad dentro de 5 años: " << edad_futura << endl;

    return 0;

}
```

### Código en Java

```
public class Main {  
  
    public static void main(String[] args) {  
  
        int edad_actual = 25;  
  
        int edad_futura = edad_actual + 5;  
  
        System.out.println("Edad dentro de 5 años: " + edad_futura);  
  
    }  
  
}
```

## Estructura de selección

Es una estructura de control que permite que se tomen decisiones mientras el programa esté corriendo. En otras palabras, para cualquier problema planteado, se presentan varios caminos, y de acuerdo a unas condiciones el programa elige un camino, dependiendo del resultado de las condiciones.

A diferencia de la estructura de secuencia, aquí el camino no siempre será lineal, sino que podrá tomar un orden diferente en la ruta. Según la condición lógica, si es verdadera o falsa, se ejecuta una u otra línea o bloque de código.

De acuerdo al lenguaje de programación en que se esté trabajando existen unos tipos de selección (pueden variar de un lenguaje a otro), de los cuales se comparten los más usados:

- **If simple.** Ejecuta un bloque si y solo si la condición es verdadera. Se ejecuta la condición con una sola acción.
- **If-else.** Cuando la condición es verdadera ejecuta un bloque; cuando es falsa, ejecuta un bloque diferente. Se usa cuando hay dos caminos alternativos.
- **If-elif-else (Python) / else if (C++, Java).** Permite elegir entre varias condiciones. De acuerdo a la que se cumpla, ejecuta un bloque exclusivo. Se emplea cuando hay más de dos condiciones posibles.
- **Switch-case (C++, Java).** Maneja múltiples opciones según el valor de una variable, ejecutando el bloque que corresponda. Se recomienda cuando existen opciones fijas, como en un menú.

Para conocer como se ve la estructura de selección, se comparten unos ejemplos:

#### Código en C++ de if simple

```
#include <iostream>

using namespace std;

int main() {

    int numero;

    cout << "Ingrese un número: ";

    cin >> numero;

    if (numero > 0)
```

```
    cout << "El número es positivo." << endl;

    return 0;

}
```

### **Código en Java de if else**

```
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.print("Ingrese un número: ");

        int numero = sc.nextInt();

        if (numero % 2 == 0) {

            System.out.println("El número es par.");

        } else {

            System.out.println("El número es impar.");

        }

    }

}
```

## Código en C++ de else if

```
#include <iostream>

using namespace std;

int main() {
    float nota;
    cout << "Ingrese la nota: ";
    cin >> nota;

    if (nota >= 4.5)
        cout << "Excelente" << endl;
    else if (nota >= 3.5)
        cout << "Bueno" << endl;
    else if (nota >= 3.0)
        cout << "Aceptable" << endl;
    else
        cout << "Reprobado" << endl;
    return 0;
}
```

## Código en Java de Switch-case

```
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.println("1. Sumar\n2. Restar\n3. Salir");

        System.out.print("Ingrese una opción: ");

        int opcion = sc.nextInt();

        switch (opcion) {

            case 1:

                System.out.println("Has elegido Sumar.");

                break;

            case 2:

                System.out.println("Has elegido Restar.");

                break;

            case 3:

                System.out.println("Has elegido Salir.");

                break;

            default:

                System.out.println("Opción no válida.");

        }

    }

}
```



## Estructura de iteración

La iteración, también conocida como ciclo o bucle, es una estructura de control utilizada para automatizar procesos repetitivos, y su finalidad es permitir la repetición de un bloque de instrucciones mientras se esté cumpliendo una condición lógica. Cuando la condición deja de cumplirse, el flujo del programa sale del bucle y sigue con el resto de instrucciones si las hay, de lo contrario finaliza. La iteración es un proceso clave de mucha ayuda para la solución de problemas donde se hace necesaria la repetición de acciones, por ejemplo:

- Procesar listas o series de datos.
- Repetir operaciones matemáticas.
- Crear menús interactivos.
- Realizar cálculos acumulativos.

De acuerdo al lenguaje de programación en que se esté trabajando existen unos tipos de iteración (pueden variar de un lenguaje a otro), de los cuales se comparten los más usados:

- **While**. La condición se evalúa antes de cada repetición y, si se cumple, el bloque de código se repite mientras siga siendo verdadera. Se puede usar cuando no se sabe cuántas veces se ejecutarán las instrucciones.
- **Do-while (C++, Java)**. Funciona como el while, pero la condición se evalúa después de la primera ejecución, lo que garantiza que el bloque se ejecuta al menos una vez. Se utiliza cuando se requiere que las instrucciones se realicen al menos una vez.

- **For.** La repetición ocurre un número determinado de veces, controlado por un contador. Se usa cuando se conoce de antemano cuántas veces se desea repetir la acción.

Para conocer como se ve la estructura de selección, analice los siguientes ejemplos:

#### Código en C++ de while

```
#include <iostream>
using namespace std;
int main() {
    int i = 1;
    while (i <= 5) {
        cout << "Número: " << i << endl;
        i++;
    }
    return 0;
}
```

### Código en Java de do while

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int opcion;
        do {
            System.out.println("1. Opción A\n2. Opción B\n3. Salir");
            System.out.print("Elija una opción: ");
            opcion = sc.nextInt();
        } while (opcion != 3);
        System.out.println("Programa finalizado.");
    }
}
```

### Código en C++ de else if

```
#include <iostream>
using namespace std;
int main() {
    float nota;
    cout << "Ingrese la nota: ";
    cin >> nota;
```

```
if (nota >= 4.5)
    cout << "Excelente" << endl;
else if (nota >= 3.5)
    cout << "Bueno" << endl;
else if (nota >= 3.0)
    cout << "Aceptable" << endl;
else
    cout << "Reprobado" << endl;
return 0;
}
```

#### **Código en Python del for**

```
suma = 0
for i in range(1, 11):
    suma += i
print("La suma es:", suma)
```

## **2.4. Funciones y modularidad**

Uno de los principios de la programación estructurada es dividir el código de un programa en partes más pequeñas, que a la vez sean independientes y reutilizables; esto se logra con funciones (conocidas también como subrutinas o procedimientos), las

cuales permiten aplicar el principio de modularidad. Cuando se logra entender el concepto de funciones, será más fácil para el programador desarrollar y mantener sus programas. Ahondando en lo conceptos se puede decir que:

- **Funciones.** Bloque de código que realiza una operación particular dentro de un programa. Puede incluir parámetros de entrada y devolver un valor como salida. Facilitan el encapsulamiento de operaciones comunes en un bloque reutilizable. Se recomienda usar nombres claros y nemotécnicos para identificar fácilmente la función (Microsoft, 2023).
- **Modularidad.** Práctica de dividir un programa en módulos más pequeños, cada uno con funciones y responsabilidades claras. Esta descomposición permite una mejor gestión, mantenimiento y comprensión del programa, en contraste con un código extenso y monolítico (Casero, 2023).

Una función puede recibir unos datos de entrada (parámetros), puede devolver un resultado (valor de retorno) y puede ser reutilizada en diferentes partes de un programa, para ello después de definida, solo tiene que ser invocada y podrá cumplir con el objetivo o la tarea para la cual fue creada.

La estructura básica de una función se compone generalmente por:

- Nombre nemotécnico o descriptivo.
- Parámetros de entrada (opcionales)
- Valor de retorno (opcional)
- Bloque de instrucciones o líneas de código, las cuales son las que realizan la tarea para la cual fue creada.

Así es la estructura de una función:

```
Función nombre_función(parámetros) → valor_retorno
```

```
    instrucciones
```

```
    return valor_retorno
```

Y aunque ya se han compartido ejemplos de funciones en anteriores códigos, se comparten ejemplos específicos de funciones en diversos lenguajes de programación:

### **Función en Python**

```
def calcular_area_circulo(radio):
```

```
    return 3.1416 * radio * radio
```

```
# Uso de la función
```

```
r = float(input("Ingrese el radio: "))
```

```
area = calcular_area_circulo(r)
```

```
print("El área del círculo es:", area)
```

## Función en C++

```
#include <iostream>

using namespace std;

// Definición de la función

double calcular_area_circulo(double radio) {

    return 3.1416 * radio * radio;

}

int main() {

    double r;

    cout << "Ingrese el radio: ";

    cin >> r;

    double area = calcular_area_circulo(r);

    cout << "El área del círculo es: " << area << endl;

    return 0;

}
```

## Función en Java

```
import java.util.Scanner;

public class Main {

    // Definición de la función

    public static double calcularAreaCirculo(double radio) {

        return 3.1416 * radio * radio;

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.print("Ingrese el radio: ");

        double r = sc.nextDouble();

        double area = calcularAreaCirculo(r);

        System.out.println("El área del círculo es: " + area);

    }

}
```

Después de conocer los ejemplos anteriores, hay que tener en cuenta que las funciones no se comportan igual, es decir, algunas pueden devolver un resultado, otras reciben datos y otras simplemente realizan la tarea para la cual fueron creadas.



Conocer los diferentes tipos de funciones, le permite a los programadores construir programas mejor organizados. A continuación, la siguiente tabla resume los tipos de funciones que se pueden utilizar:

- **Función sin parámetro y sin retorno.** Solo ejecuta las instrucciones o la tarea encomendada. Ejemplo: mostrar un mensaje.
- **Función con parámetros, pero sin retorno.** Procesa los datos, pero no devuelve el valor o el resultado. Ejemplo: impresión de un reporte.
- **Función sin parámetros, pero con retorno.** Devuelve un valor fijo. Ejemplo: obtener la fecha actual.
- **Función con parámetros y con retorno.** Procesa los datos y devuelve un valor o un resultado. Ejemplo: calcular el área cuadrada de un cultivo.

Tal como se ha podido evidenciar, las funciones permiten organizar mejor el código de un programa y hacerlo más robusto con su reutilización. En el caso de la modularidad, en el siguiente cuadro se aprecia cómo se puede ver un programa organizado de esta forma:

Ejemplo de código que muestra la modularidad en la programación estructurada
<pre>def leer_numero():     return float(input("Ingrese un número: "))  def sumar(a, b):     return a + b</pre>

```
def mostrar_resultado(resultado):  
    print("La suma es:", resultado)  
  
# Programa principal  
num1 = leer_numero()  
num2 = leer_numero()  
resultado = sumar(num1, num2)  
mostrar_resultado(resultado)
```

## 2.5. Ejemplos

En este apartado se comparten unos ejemplos de código generado en Python con ChatGPT y corriendo en Visual Studio Code. Lo primero que se hizo fue darle un prompt general para que supiera como iba a trabajar:

### Prompt inicial

Necesito que actúes como un experto en programación, pero que tengas en cuenta los siguientes parámetros para trabajar:

- Lenguaje de programación a trabajar: Python.
- Paradigma de programación: programación estructurada.
- Recursos a utilizar para el desarrollo: funciones y estructuras de control.
- Lógica de programación: lo más sencillo posible, ya que el público objetivo apenas está aprendiendo a programar.

- Cuando estés listo me dices para irte dando los problemas a resolver por medio de los programas que vas a generar.

Luego se le fueron dando cada uno de los prompts con el que generaría los códigos para los programas solicitados.

## Ejemplo 1: operación aritmética

**Prompt:** Necesito un programa que pregunte qué operación aritmética se desea realizar, solicite dos números, y con ellos haga la operación y muestre como resultado: la operación y el valor resultante de la operación.

### Código generado en Python con ChatGPT

```
# Función para realizar la operación seleccionada

def calculadora_simple():

    print("Bienvenido a la calculadora básica.")

    # Mostrar opciones

    print("\nElige la operación que deseas realizar:")

    print("1. Suma")

    print("2. Resta")

    print("3. Multiplicación")

    print("4. División")

    opcion = input("Ingresa el número de la opción elegida (1-4): ")
```

```
# Pedir los dos números

num1 = float(input("Ingresa el primer número: "))
num2 = float(input("Ingresa el segundo número: "))


# Realizar la operación y mostrar la operación completa con el resultado

if opcion == "1":

    resultado = num1 + num2

    print(f"{num1} + {num2} = {resultado}")

elif opcion == "2":

    resultado = num1 - num2

    print(f"{num1} - {num2} = {resultado}")

elif opcion == "3":

    resultado = num1 * num2

    print(f"{num1} * {num2} = {resultado}")

elif opcion == "4":

    if num2 != 0:

        resultado = num1 / num2

        print(f"{num1} / {num2} = {resultado}")

    else:
```

```
print("Error: no se puede dividir entre cero.")

else:

    print("Opción no válida. Intenta de nuevo.")
```

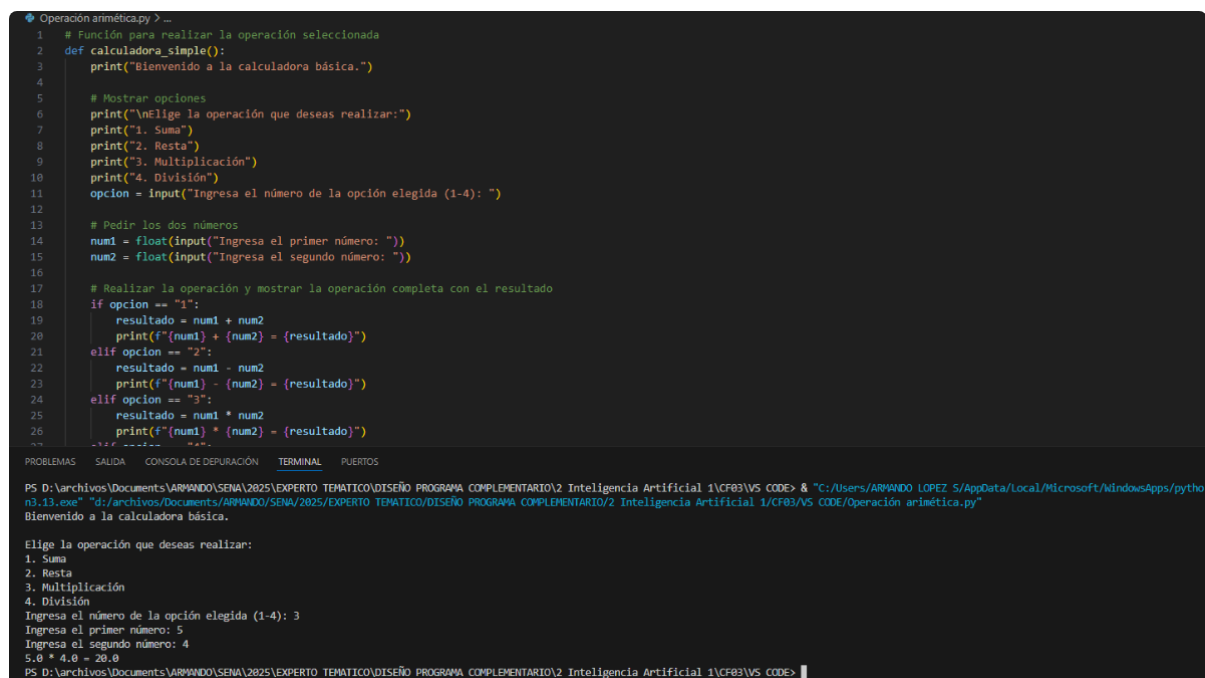
# Llamar a la función principal

calculadora\_simple()

resultado = sumar(num1, num2)

mostrar\_resultado(resultado)

**Figura 4.** Pantallazo del código en Visual Studio Code y ejecución en la terminal



```
Operación aritmética.py > ...
1 # Función para realizar la operación seleccionada
2 def calculadora_simple():
3     print("Bienvenido a la calculadora básica.")
4
5     # Mostrar opciones
6     print("\nElige la operación que deseas realizar:")
7     print("1. Suma")
8     print("2. Resta")
9     print("3. Multiplicación")
10    print("4. División")
11    opcion = input("Ingresa el número de la opción elegida (1-4): ")
12
13    # Pedir los dos números
14    num1 = float(input("Ingresa el primer número: "))
15    num2 = float(input("Ingresa el segundo número: "))
16
17    # Realizar la operación y mostrar la operación completa con el resultado
18    if opcion == "1":
19        resultado = num1 + num2
20        print(f"{num1} + {num2} = {resultado}")
21    elif opcion == "2":
22        resultado = num1 - num2
23        print(f"{num1} - {num2} = {resultado}")
24    elif opcion == "3":
25        resultado = num1 * num2
26        print(f"{num1} * {num2} = {resultado}")
27    elif opcion == "4":
28        resultado = num1 / num2
29        print(f"{num1} / {num2} = {resultado}")
30    else:
31        print("Opción no válida. Intenta de nuevo.")
32
33    # Llamar a la función principal
34    calculadora_simple()
35
36    # Mostrar resultado
37    mostrar_resultado(resultado)
38
39    # Preguntar si se desea continuar
40    continuar = input("¿Deseas continuar? (s/n): ")
41    if continuar == "n":
42        return
43
44    # Repetir el proceso
45    calculadora_simple()
46
47    # Fin del programa
48    print("Fin del programa.")
49
50 if __name__ == "__main__":
51     calculadora_simple()
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS

```
PS D:\archivos\Documents\ARWANDO\SENA\2025\EXPERTO TEMATICO\DISEÑO PROGRAMA COMPLEMENTARIO\2 Inteligencia Artificial 1\CF83\VS CODE> & "C:/Users/ARWANDO LOPEZ S/AppData/Local/Microsoft/WindowsApps/python3.13.exe" "d:/archivos/Documents/ARWANDO/SENA/2025/EXPERTO TEMATICO/DISEÑO PROGRAMA COMPLEMENTARIO/2 Inteligencia Artificial 1/CF83/VS CODE/Operación aritmética.py"
Bienvenido a la calculadora básica.

Elige la operación que deseas realizar:
1. Suma
2. Resta
3. Multiplicación
4. División
Ingresa el número de la opción elegida (1-4): 3
Ingresa el primer número: 5
Ingresa el segundo número: 4
5.0 * 4.0 = 20.0
PS D:\archivos\Documents\ARWANDO\SENA\2025\EXPERTO TEMATICO\DISEÑO PROGRAMA COMPLEMENTARIO\2 Inteligencia Artificial 1\CF83\VS CODE>
```

## Ejemplo 2: calculadora

**Prompt:** Un estudiante al que se le dañó su calculadora, desea hacer un programa que le permita realizar las 4 operaciones aritméticas.

### Código generado en Python con ChatGPT

```
# Función para sumar dos números

def sumar(a, b):

    return a + b


# Función para restar dos números

def restar(a, b):

    return a - b


# Función para multiplicar dos números

def multiplicar(a, b):

    return a * b


# Función para dividir dos números

def dividir(a, b):

    if b != 0:

        return a / b
```

```
else:
```

```
    return "Error: no se puede dividir entre cero."
```

```
# Función principal para interactuar con el usuario
```

```
def calculadora():
```

```
    print("Calculadora Básica")
```

```
    print("Operaciones disponibles: suma, resta, multiplicación, división\n")
```

```
# Pedir los dos números
```

```
    num1 = float(input("Ingrese el primer número: "))
```

```
    num2 = float(input("Ingrese el segundo número: "))
```

```
# Pedir la operación a realizar
```

```
    print("\nElige la operación a realizar:")
```

```
    print("1. Suma")
```

```
    print("2. Resta")
```

```
    print("3. Multiplicación")
```

```
    print("4. División")
```

```
    opcion = input("Ingresa el número de la opción elegida (1-4): ")
```

```
# Realizar la operación seleccionada

if opcion == "1":

    resultado = sumar(num1, num2)

    print("El resultado de la suma es:", resultado)


elif opcion == "2":

    resultado = restar(num1, num2)

    print("El resultado de la resta es:", resultado)


elif opcion == "3":

    resultado = multiplicar(num1, num2)

    print("El resultado de la multiplicación es:", resultado)


elif opcion == "4":

    resultado = dividir(num1, num2)

    print("El resultado de la división es:", resultado)


else:

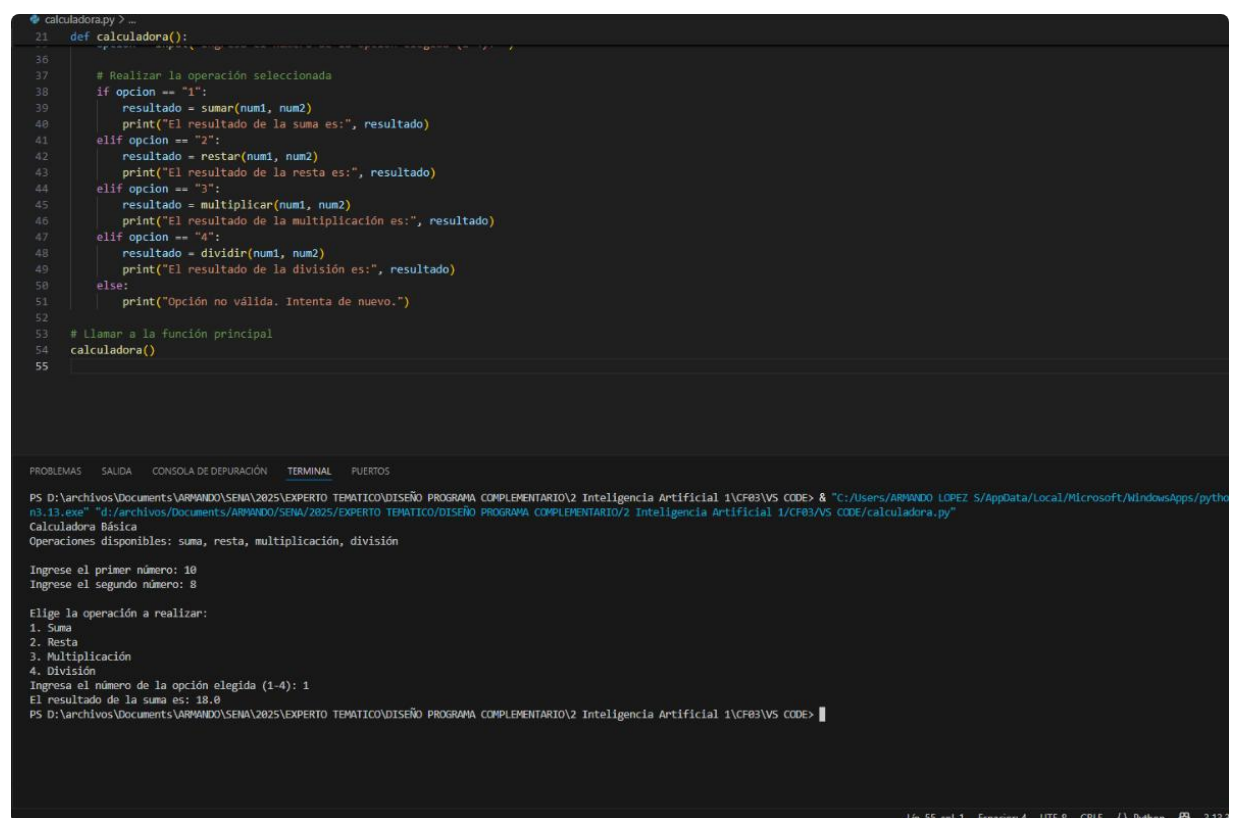
    print("Opción no válida. Intenta de nuevo.")
```



```
# Llamar a la función principal
```

```
calculadora()
```

**Figura 5.** Pantallazo del código en Visual Studio Code y ejecución en la terminal.



```
21 def calculadora():
22     # Realizar la operación seleccionada
23     if opcion == "1":
24         resultado = sumar(num1, num2)
25         print("El resultado de la suma es:", resultado)
26     elif opcion == "2":
27         resultado = restar(num1, num2)
28         print("El resultado de la resta es:", resultado)
29     elif opcion == "3":
30         resultado = multiplicar(num1, num2)
31         print("El resultado de la multiplicación es:", resultado)
32     elif opcion == "4":
33         resultado = dividir(num1, num2)
34         print("El resultado de la división es:", resultado)
35     else:
36         print("Opción no válida. Intenta de nuevo.")
37
38 # Llamar a la función principal
39 calculadora()
40
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS

```
PS D:\Archivos\Documents\ARMANDO\SENA\2025\EXPERTO TEMATICO\DISEÑO PROGRAMA COMPLEMENTARIO\2 Inteligencia Artificial 1\CFB3\VS CODE> & "C:/Users/ARMANDO LOPEZ S/AppData/Local/Microsoft/WindowsApps/python3.13.exe" "d:/archivos/Documents/ARMANDO/SENA/2025/EXPERTO TEMATICO/DISEÑO PROGRAMA COMPLEMENTARIO/2 Inteligencia Artificial 1/CFB3/VS CODE/calculadora.py"
Calculadora Básica
Operaciones disponibles: suma, resta, multiplicación, división

Ingrese el primer número: 10
Ingrese el segundo número: 8

Elige la operación a realizar:
1. Suma
2. Resta
3. Multiplicación
4. División
Ingresa el número de la opción elegida (1-4): 1
El resultado de la suma es: 18.0
PS D:\Archivos\Documents\ARMANDO\SENA\2025\EXPERTO TEMATICO\DISEÑO PROGRAMA COMPLEMENTARIO\2 Inteligencia Artificial 1\CFB3\VS CODE>
```

## Ejemplo 3: lista ordenada

**Prompt:** Juanito desea ayudar a su profesora, quien ha tomado los nombres de los 10 estudiantes que han ingresado por primera vez a su clase; para ello Juanito necesita crear un programa donde escriba los 10 nombres y se los entregue en orden alfabético ascendente.

### Código generado en Python con ChatGPT

# Función para registrar y ordenar los nombres de los estudiantes

```
def ordenar_nombres_estudiantes():
```

```
    # Lista para guardar los nombres
```

```
    lista_nombres = []
```

```
    # Pedir los 10 nombres
```

```
    for i in range(1, 11):
```

```
        nombre = input(f"Ingrese el nombre del estudiante {i}: ")
```

```
        lista_nombres.append(nombre)
```

```
    # Ordenar la lista alfabéticamente en orden ascendente
```

```
    lista_nombres.sort()
```

```
    # Mostrar la lista ordenada
```

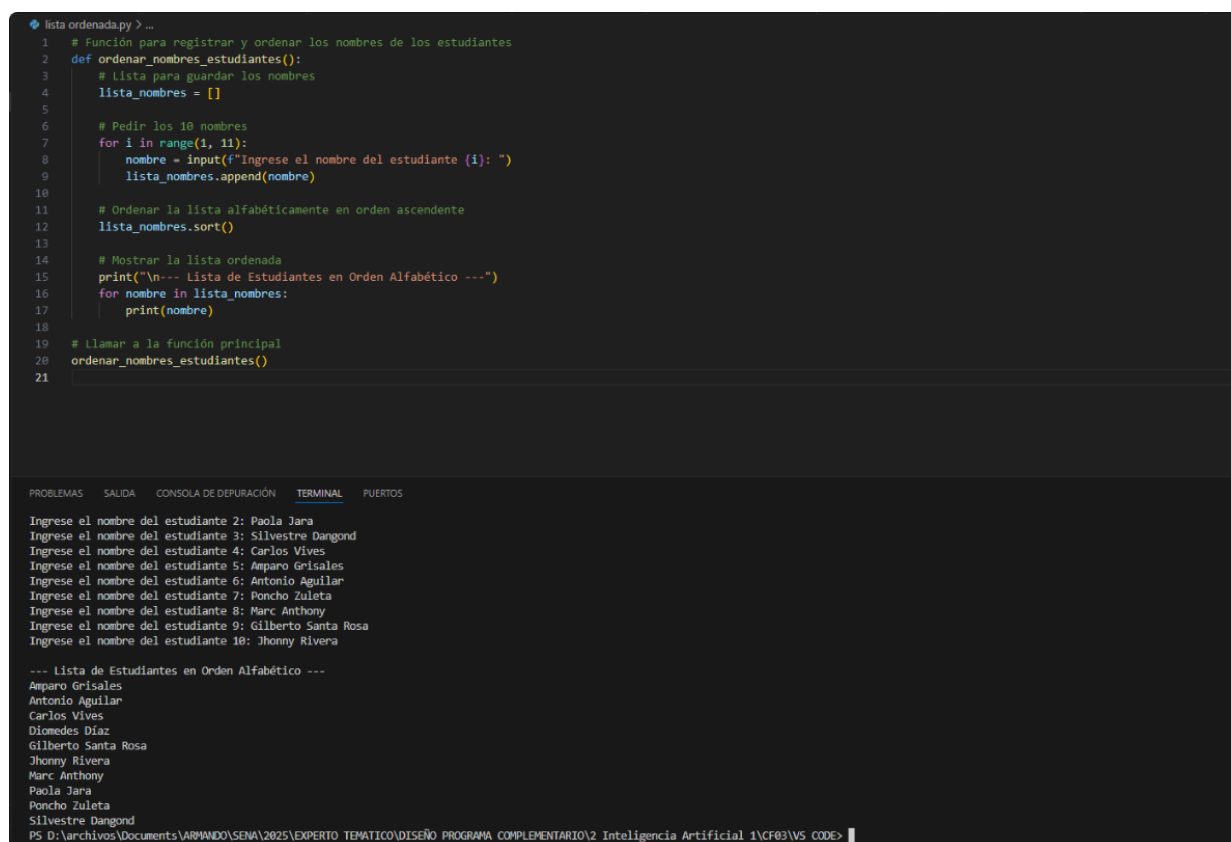
```
    print("\n--- Lista de Estudiantes en Orden Alfabético ---")
```

```
    for nombre in lista_nombres:
```

```
        print(nombre)
```

```
# Llamar a la función principal  
ordenar_nombres_estudiantes()
```

**Figura 6.** Pantallazo del código en Visual Studio Code y ejecución en la terminal.



```
lista_ordenada.py > ...  
1 # Función para registrar y ordenar los nombres de los estudiantes  
2 def ordenar_nombres_estudiantes():  
3     # Lista para guardar los nombres  
4     lista_nombres = []  
5  
6     # Pedir los 10 nombres  
7     for i in range(1, 11):  
8         nombre = input(f"Ingrese el nombre del estudiante {i}: ")  
9         lista_nombres.append(nombre)  
10  
11     # Ordenar la lista alfabéticamente en orden ascendente  
12     lista_nombres.sort()  
13  
14     # Mostrar la lista ordenada  
15     print("\n--- Lista de Estudiantes en Orden Alfabético ---")  
16     for nombre in lista_nombres:  
17         print(nombre)  
18  
19 # Llamar a la función principal  
20 ordenar_nombres_estudiantes()  
21
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS

```
Ingrese el nombre del estudiante 2: Paola Jara  
Ingrese el nombre del estudiante 3: Silvestre Dangond  
Ingrese el nombre del estudiante 4: Carlos Vives  
Ingrese el nombre del estudiante 5: Amparo Grisales  
Ingrese el nombre del estudiante 6: Antonio Aguilar  
Ingrese el nombre del estudiante 7: Poncho Zuleta  
Ingrese el nombre del estudiante 8: Marc Anthony  
Ingrese el nombre del estudiante 9: Gilberto Santa Rosa  
Ingrese el nombre del estudiante 10: Jhonny Rivera  
  
--- Lista de Estudiantes en Orden Alfabético ---  
Amparo Grisales  
Antonio Aguilar  
Carlos Vives  
Diomedes Díaz  
Gilberto Santa Rosa  
Jhonny Rivera  
Marc Anthony  
Paola Jara  
Poncho Zuleta  
Silvestre Dangond  
PS D:\archivos\Documents\ARMANDO\SENA\2025\EXPERTO TEMATICO\DISEÑO PROGRAMA COMPLEMENTARIO\2 Inteligencia Artificial 1\CF03\VS CODE>
```

## Ejemplo 4: promedio de notas

Prompt: un estudiante está cursando 3 materias, cada materia es calificada con 3 notas, donde la nota 1 equivale al 30 %, la nota 2 al 30 % y la nota 3 al 40 % del total del curso. Se necesita obtener el promedio por materia y el promedio general del estudiante.

### Código generado en Python con ChatGPT

# Función para calcular el promedio ponderado de una materia

```
def promedio_materia():
```

```
    print("\nIngresa las notas de la materia:")
```

```
    nota1 = float(input("Nota 1 (30%): "))
```

```
    nota2 = float(input("Nota 2 (30%): "))
```

```
    nota3 = float(input("Nota 3 (40%): "))
```

# Calcular el promedio ponderado

```
    promedio = (nota1 * 0.3) + (nota2 * 0.3) + (nota3 * 0.4)
```

```
    return promedio
```

# Función principal para manejar el proceso

```
def promedio_general():
```

```
    total_promedios = 0
```

# Calcular el promedio para las 3 materias

```
    for i in range(1, 4):
```

```
        print("\n--- Materia", i, "---")
```

```
promedio = promedio_materia()

print("Promedio de la materia", i, ":", promedio)

total_promedios += promedio
```

```
# Calcular el promedio general
```

```
promedio_general = total_promedios / 3

print("\n--- Promedio General del Estudiante ---")

print("Promedio general:", promedio_general)
```

```
# Llamar a la función principal
```

```
promedio_general()
```

```
# Llamar a la función principal
```

```
ordenar_nombres_estudiantes()
```

**Figura 7.** Pantallazo del código en Visual Studio Code y ejecución en la terminal.

```

promedio de notas.py >...
10     return promedio
11
12 # Función principal para manejar el proceso
13 def promedio_general():
14     total_promedios = 0
15
16     # Calcular el promedio para las 3 materias
17     for i in range(1, 4):
18         print("\n--- Materia", i, "---")
19         promedio = promedio_materia()
20         print("Promedio de la materia", i, ":", promedio)
21         total_promedios += promedio
22
23     # Calcular el promedio general
24     promedio_general = total_promedios / 3
25     print("\n--- Promedio General del Estudiante ---")
26     print("Promedio general:", promedio_general)
27
28 # Llamar a la función principal
29 promedio_general()
30

```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS

```

Nota 3 (40%): 3
Promedio de la materia 1 : 3.6
--- Materia 2 ---
Ingresa las notas de la materia:
Nota 1 (30%): 5
Nota 2 (30%): 5
Nota 3 (40%): 2
Promedio de la materia 2 : 3.8
--- Materia 3 ---
Ingresa las notas de la materia:
Nota 1 (30%): 4
Nota 2 (30%): 3
Nota 3 (40%): 5
Promedio de la materia 3 : 4.1
--- Promedio General del Estudiante ---
Promedio general: 3.8333333333333335
PS D:\archivos\Documents\ARMANDO\SENA\2025\EXPERTO TEMATICO\DISEÑO PROGRAMA COMPLEMENTARIO\2 Inteligencia Artificial 1\CF03\VS CODE>

```

## Ejemplo 5: venta de productos electrónicos

**Prompt:** Ricardo está vendiendo dispositivos electrónicos. Carlos le ha hecho una compra y necesita hacerle la factura. Para hacer la factura necesita saber la cantidad de dispositivos vendidos, el valor del dispositivo vendido, el descuento por la venta, el impuesto por la venta y el total a pagar en la factura

### Código generado en Python con ChatGPT

# Función para calcular la factura de la venta

```
def calcular_factura():
```

```
# Pedir la cantidad de dispositivos vendidos

cantidad = int(input("Cantidad de dispositivos vendidos: "))


# Pedir el valor unitario de cada dispositivo

valor_unitario = float(input("Valor de cada dispositivo: "))


# Calcular el subtotal

subtotal = cantidad * valor_unitario


# Pedir el descuento (en porcentaje)

descuento = float(input("Descuento en porcentaje (%): "))


# Calcular el monto de descuento

monto_descuento = subtotal * (descuento / 100)


# Calcular el subtotal después del descuento

subtotal_con_descuento = subtotal - monto_descuento


# Pedir el impuesto (en porcentaje)

impuesto = float(input("Impuesto en porcentaje (%): "))
```

```
# Calcular el monto de impuesto

monto_impuesto = subtotal_con_descuento * (impuesto / 100)


# Calcular el total a pagar

total_pagar = subtotal_con_descuento + monto_impuesto


# Mostrar la factura

print("\n--- Factura de Compra ---")

print("Cantidad de dispositivos:", cantidad)

print("Valor unitario:", valor_unitario)

print("Subtotal:", subtotal)

print("Descuento aplicado:", monto_descuento)

print("Subtotal con descuento:", subtotal_con_descuento)

print("Impuesto aplicado:", monto_impuesto)

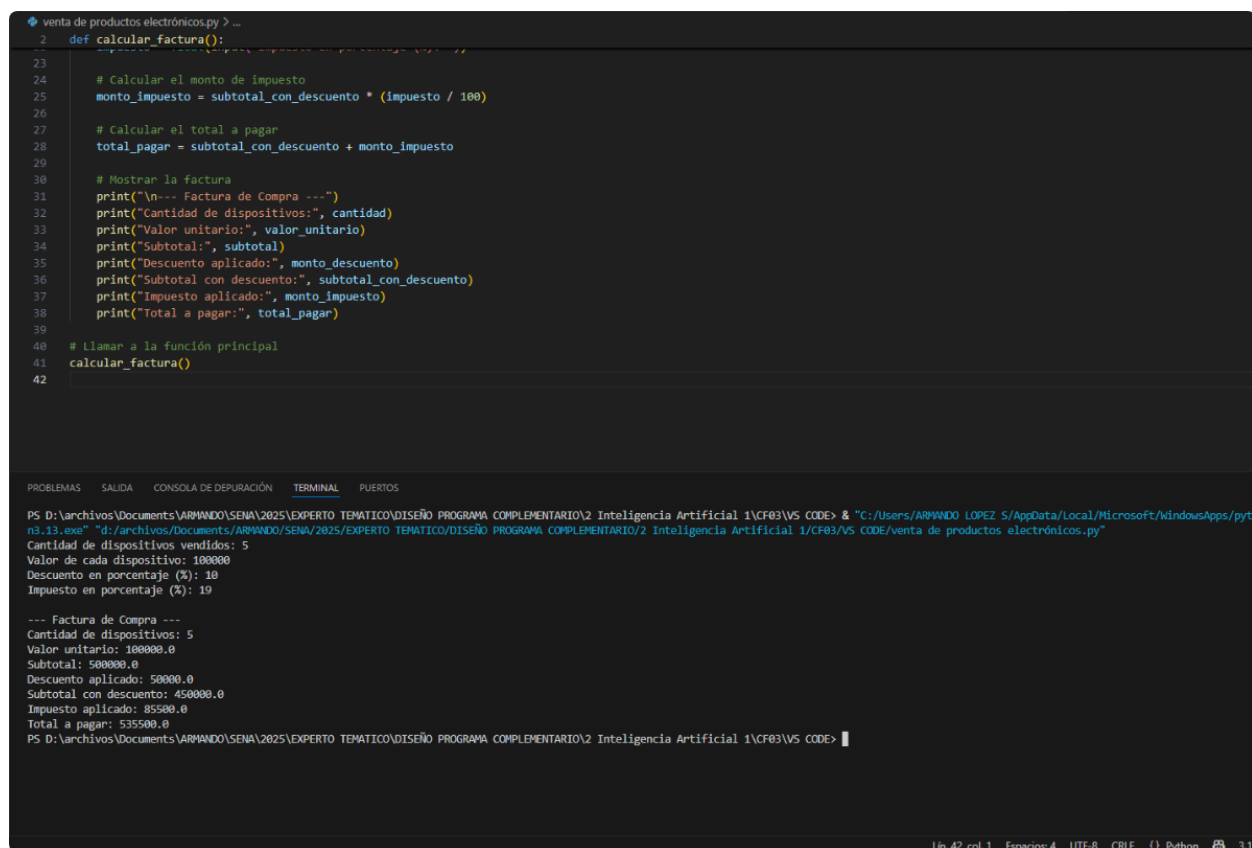
print("Total a pagar:", total_pagar)


# Llamar a la función principal

calcular_factura()
```



**Figura 8.** Pantallazo del código en Visual Studio Code y ejecución en la terminal.



```

2  def calcular_factura():
23
24     # Calcular el monto de impuesto
25     monto_impuesto = subtotal_con_descuento * (impuesto / 100)
26
27     # Calcular el total a pagar
28     total_pagar = subtotal_con_descuento + monto_impuesto
29
30     # Mostrar la factura
31     print("\n--- Factura de Compra ---")
32     print("Cantidad de dispositivos:", cantidad)
33     print("Valor unitario:", valor_unitario)
34     print("Subtotal:", subtotal)
35     print("Descuento aplicado:", monto_descuento)
36     print("Subtotal con descuento:", subtotal_con_descuento)
37     print("Impuesto aplicado:", monto_impuesto)
38     print("Total a pagar:", total_pagar)
39
40 # Llamar a la función principal
41 calcular_factura()
42

```

```

PS D:\archivos\Documents\ARMANDO\SENA\2025\EXPERTO TEMATICO\DISEÑO PROGRAMA COMPLEMENTARIO\2 Inteligencia Artificial 1\CF03\VS CODE> & "C:/Users/ARMANDO LOPEZ S/AppData/Local/Microsoft/WindowsApps/python3.11.exe" "d:/archivos/Documents/ARMANDO/SENA/2025/EXPERTO TEMATICO/DISEÑO PROGRAMA COMPLEMENTARIO/2 Inteligencia Artificial 1/CF03/VS CODE/venta de productos electronicos.py"
Cantidad de dispositivos vendidos: 5
Valor de cada dispositivos 100000
Descuento en porcentaje (%): 10
Impuesto en porcentaje (%): 19

--- Factura de Compra ---
Cantidad de dispositivos: 5
Valor unitario: 100000.0
Subtotal: 500000.0
Descuento aplicado: 50000.0
Subtotal con descuento: 450000.0
Impuesto aplicado: 85500.0
Total a pagar: 535500.0
PS D:\archivos\Documents\ARMANDO\SENA\2025\EXPERTO TEMATICO\DISEÑO PROGRAMA COMPLEMENTARIO\2 Inteligencia Artificial 1\CF03\VS CODE>

```

## Ejemplo 6: venta de leche

**Prompt:** en la finca La Costa, todos los días se está vendiendo la leche a la empresa la Pradera. El valor del litro de leche es \$2.000. La leche se paga quincenal. Se necesita saber la cantidad de litros producidos desde el día 1 al día 15. Se necesita saber el total de leche vendida y el total de dinero recibido por venta de leche.

### Código generado en Python con ChatGPT

```
# Función para calcular la venta de leche en 15 días

def venta_leche_quincenal():

    # Precio del litro de leche

    precio_litro = 2000

    # Inicializar el total de litros

    total_litros = 0

    # Pedir la cantidad de litros de leche producidos cada día (día 1 al 15)

    for dia in range(1, 16):

        litros = float(input(f"Ingrese los litros de leche del día {dia}: "))

        total_litros += litros

    # Calcular el total de dinero recibido

    total_dinero = total_litros * precio_litro
```

## # Mostrar los resultados

```
print("\n--- Resumen de Venta Quincenal ---")

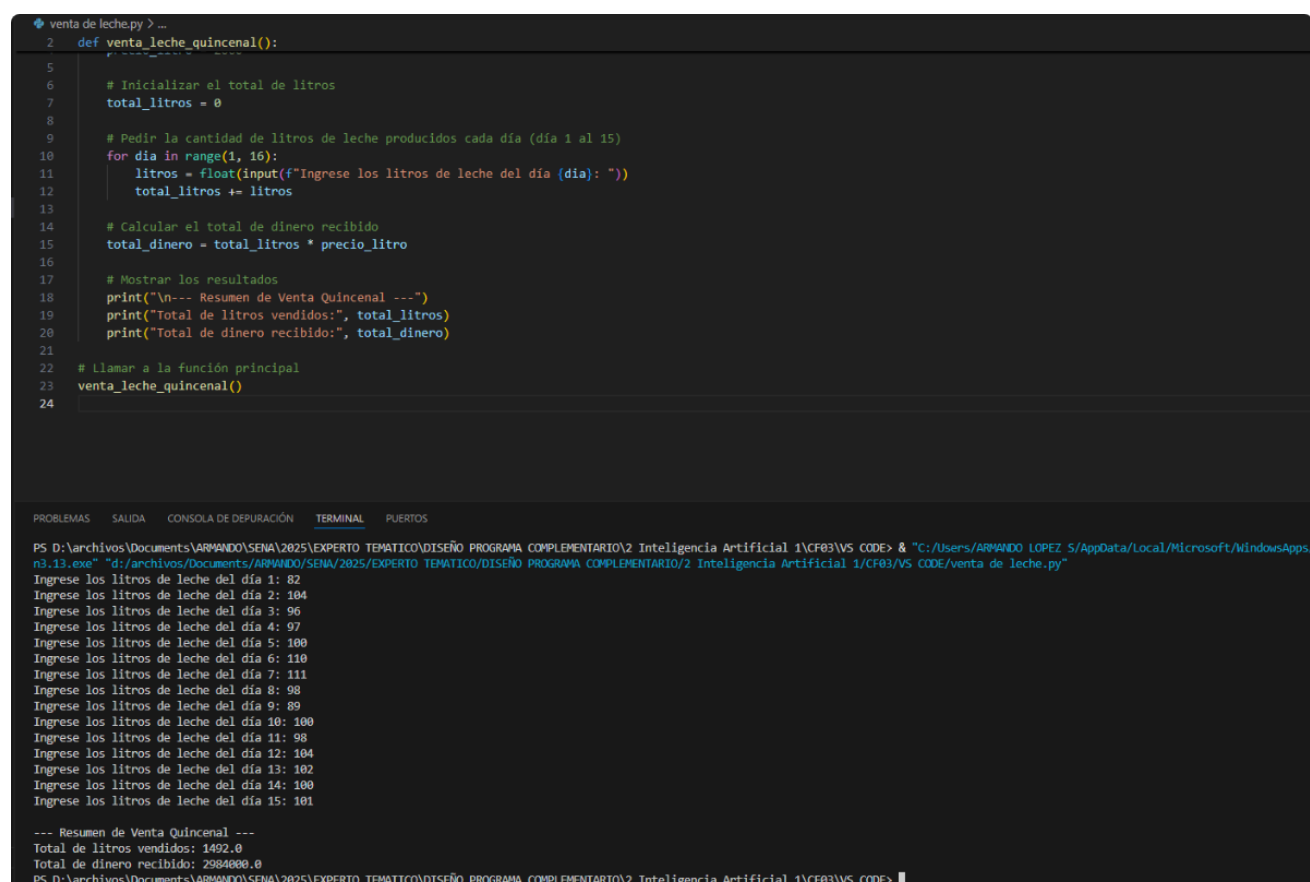
print("Total de litros vendidos:", total_litros)

print("Total de dinero recibido:", total_dinero)
```

## # Llamar a la función principal

```
venta_leche_quincenal()
```

**Figura 9.** Pantallazo del código en Visual Studio Code y ejecución en la terminal.



```

venta de leche.py > ...
2 def venta_leche_quincenal():
3     # Inicializar el total de litros
4     total_litros = 0
5
6     # Pedir la cantidad de litros de leche producidos cada día (día 1 al 15)
7     for dia in range(1, 16):
8         litros = float(input(f"Ingrese los litros de leche del día {dia}: "))
9         total_litros += litros
10
11     # Calcular el total de dinero recibido
12     total_dinero = total_litros * precio_litro
13
14     # Mostrar los resultados
15     print("\n--- Resumen de Venta Quincenal ---")
16     print("Total de litros vendidos:", total_litros)
17     print("Total de dinero recibido:", total_dinero)
18
19 # Llamar a la función principal
20 venta_leche_quincenal()
21
22
23
24

```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS

```

PS D:\archivos\Documents\ARMANDO\SENA\2025\EXPERTO TEMATICO\DISEÑO PROGRAMA COMPLEMENTARIO\2 Inteligencia Artificial 1\CF03\VS CODE> & "c:/Users/ARMANDO LOPEZ S/AppData/Local/Microsoft/WindowsApps/python3.13.exe" "d:/archivos/Documents/ARMANDO/SENA/2025/EXPERTO TEMATICO/DISEÑO PROGRAMA COMPLEMENTARIO/2 Inteligencia Artificial 1/CF03/VS CODE/venta de leche.py"
Ingrese los litros de leche del día 1: 82
Ingrese los litros de leche del día 2: 104
Ingrese los litros de leche del día 3: 96
Ingrese los litros de leche del día 4: 97
Ingrese los litros de leche del día 5: 100
Ingrese los litros de leche del día 6: 110
Ingrese los litros de leche del día 7: 111
Ingrese los litros de leche del día 8: 98
Ingrese los litros de leche del día 9: 80
Ingrese los litros de leche del día 10: 100
Ingrese los litros de leche del día 11: 98
Ingrese los litros de leche del día 12: 104
Ingrese los litros de leche del día 13: 102
Ingrese los litros de leche del día 14: 100
Ingrese los litros de leche del día 15: 101

--- Resumen de Venta Quincenal ---
Total de litros vendidos: 1492.0
Total de dinero recibido: 2984000.0
PS D:\archivos\Documents\ARMANDO\SENA\2025\EXPERTO TEMATICO\DISEÑO PROGRAMA COMPLEMENTARIO\2 Inteligencia Artificial 1\CF03\VS CODE>

```

## Ejemplo 7: entrenamiento atlético

**Prompt:** un atleta aficionado se está preparando para la media maratón de la independencia realizada en Valledupar. El correrá en la categoría de 5 kilómetros. El récord histórico está en 25 minutos. Después de un entrenamiento riguroso, él está haciendo el recorrido por 5 días, en donde lleva una libreta y anota el tiempo que se toma en el recorrido de los 5 kilómetros, y quiere saber cuánto le falta por día para batir el récord histórico. También desea saber el promedio de tiempo que se ha tomado en el recorrido de los 5 días.

### Código generado en Python con ChatGPT

```
# Función para calcular las diferencias con el récord y el promedio

def entrenamiento_atleta():

    # Récord histórico en minutos

    record = 25

    # Inicializar variables

    total_tiempo = 0

    # Pedir los tiempos para cada uno de los 5 días

    for dia in range(1, 6):

        tiempo = float(input(f"Ingrese el tiempo del día {dia} (en minutos): "))
```

```
diferencia = tiempo - record

if diferencia > 0:

    print(f"El atleta necesita mejorar {diferencia} minutos para igualar el récord.")

elif diferencia == 0:

    print("¡El atleta igualó el récord!")

else:

    print(f"¡El atleta superó el récord por {-diferencia} minutos!")


# Acumular el tiempo para el promedio

total_tiempo += tiempo


# Calcular el promedio de tiempo

promedio = total_tiempo / 5


# Mostrar el promedio

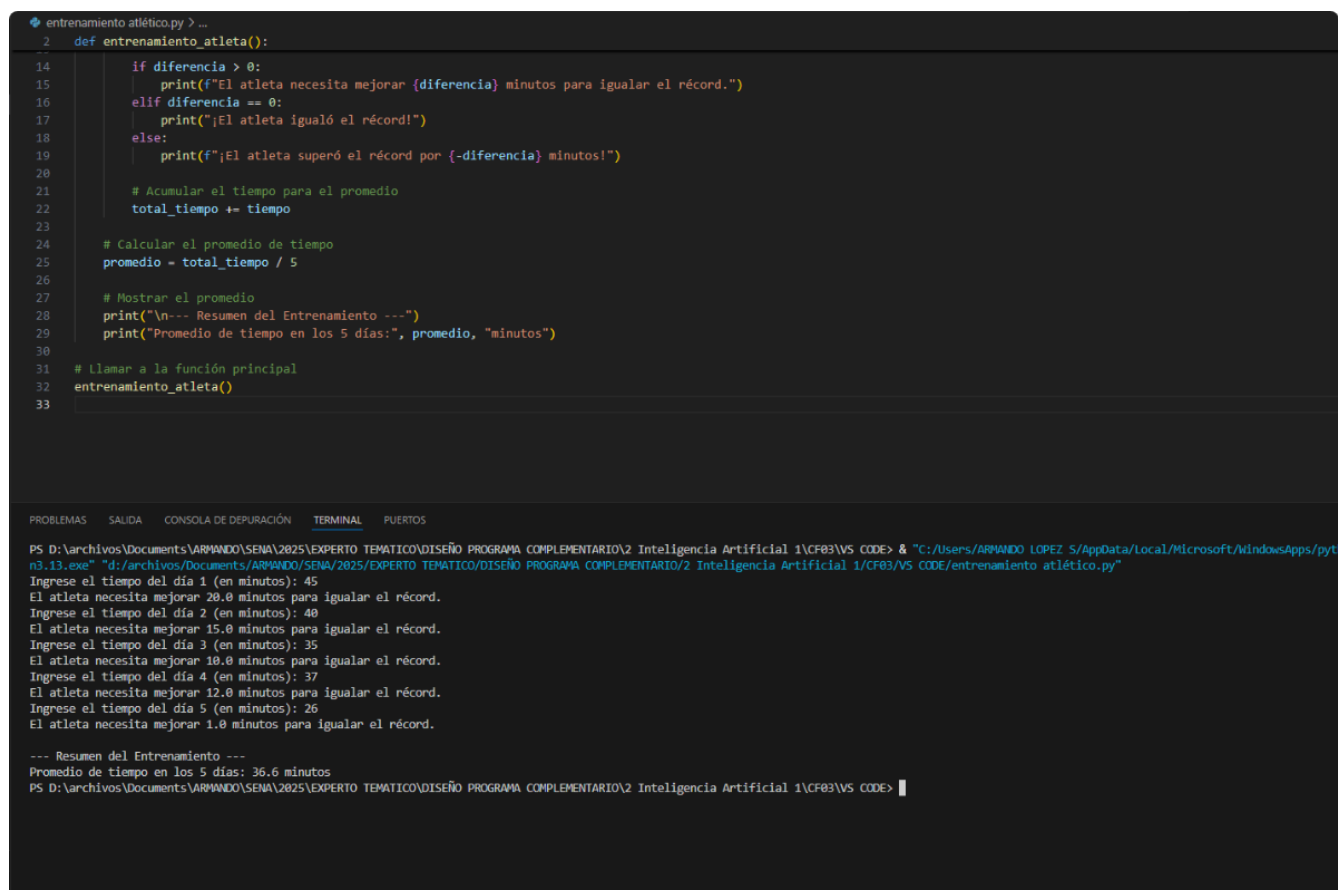
print("\n--- Resumen del Entrenamiento ---")

print("Promedio de tiempo en los 5 días:", promedio, "minutos")


# Llamar a la función principal

entrenamiento_atleta()
```

**Figura 10.** Pantallazo del código en Visual Studio Code y ejecución en la terminal.



```

entrenamiento atletico.py > ...
2 def entrenamiento_atleta():
14     if diferencia > 0:
15         print(f"El atleta necesita mejorar {diferencia} minutos para igualar el récord.")
16     elif diferencia == 0:
17         print("¡El atleta igualó el récord!")
18     else:
19         print(f"¡El atleta superó el récord por {-diferencia} minutos!")
20
21     # Acumular el tiempo para el promedio
22     total_tiempo += tiempo
23
24     # Calcular el promedio de tiempo
25     promedio = total_tiempo / 5
26
27     # Mostrar el promedio
28     print("\n--- Resumen del Entrenamiento ---")
29     print("Promedio de tiempo en los 5 días:", promedio, "minutos")
30
31 # Llamar a la función principal
32 entrenamiento_atleta()
33
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  PUERTOS
PS D:\archivos\Documents\ARMANDO\SENA\2025\EXPERTO TEMATICO\DISEÑO PROGRAMA COMPLEMENTARIO\2 Inteligencia Artificial 1\CF03\VS CODE> & "C:/Users/ARMANDO LOPEZ S/AppData/Local/Microsoft/WindowsApps/python3.13.exe" "d:/archivos/Documents/ARMANDO/SENA/2025/EXPERTO TEMATICO/DISEÑO PROGRAMA COMPLEMENTARIO/2 Inteligencia Artificial 1/CF03/VS CODE/entrenamiento atletico.py"
Ingrese el tiempo del día 1 (en minutos): 45
El atleta necesita mejorar 20.0 minutos para igualar el récord.
Ingrese el tiempo del día 2 (en minutos): 40
El atleta necesita mejorar 15.0 minutos para igualar el récord.
Ingrese el tiempo del día 3 (en minutos): 35
El atleta necesita mejorar 10.0 minutos para igualar el récord.
Ingrese el tiempo del día 4 (en minutos): 37
El atleta necesita mejorar 12.0 minutos para igualar el récord.
Ingrese el tiempo del día 5 (en minutos): 26
El atleta necesita mejorar 1.0 minutos para igualar el récord.
--- Resumen del Entrenamiento ---
Promedio de tiempo en los 5 días: 36.6 minutos
PS D:\archivos\Documents\ARMANDO\SENA\2025\EXPERTO TEMATICO\DISEÑO PROGRAMA COMPLEMENTARIO\2 Inteligencia Artificial 1\CF03\VS CODE>

```

Como se ha podido verificar en los diferentes ejemplos, los códigos fueron generados con la versión gratuita de ChatGPT y probados en el editor de código Visual Studio Code. Hay que tener presente que para que el código pueda correr sin problemas, deben tener instalado Python y configurado el editor. Se hace la salvedad, porque algunos intentan correr el código de cualquier lenguaje, pero no tienen instaladas las extensiones o cualquier otro recurso tanto en el editor como en el equipo para que el código pueda ser ejecutado sin ningún problema.

**Recomendación:** para las personas que desean aprender o asimilar mejor sus conocimientos, pueden intentar desarrollar el programa o en su defecto generarlo en la

IA gen y luego escribirlo a mano en una hoja, después digitarlo en el editor; esto se hace con el fin de ir estimulando su memoria muscular, lo cual le ayuda a irse familiarizando con el lenguaje de programación, ir estimulando la lógica, ir practicando y cuando menos lo piense podrá ir desenvolviéndose y codificando sus propios programas.

En el siguiente video se explorarán ejemplos de prompts que permiten generar códigos en Python con ayuda de ChatGPT.

#### **Video 4. Ejemplo1. Generación de códigos en Python con ChatGPT**



[Enlace de reproducción del video](#)

### Síntesis del video: generación de códigos en Python

En este video se presenta una demostración práctica del uso de inteligencia artificial para la generación automática de programas en lenguaje Python, utilizando prompts diseñados previamente.

Se comienza estableciendo un prompt inicial general, que define los parámetros bajo los cuales deben generarse los programas: lenguaje Python, enfoque en programación estructurada, uso de funciones y estructuras de control, y una lógica simple y clara, adecuada para principiantes. Esta estrategia permite mantener la coherencia y reducir la necesidad de repetir instrucciones en cada nuevo problema planteado.

Posteriormente, se presentan cuatro problemas específicos, cada uno con un enunciado contextualizado:

- **Operación aritmética:** solicita realizar una operación matemática entre dos números, con elección por parte del usuario.
- **Lista ordenada:** requiere ingresar y ordenar alfabéticamente una lista de nombres.
- **Venta de leche:** calcula el total de leche vendida en una quincena y el ingreso generado.

Entrenamiento atlético: analiza los tiempos de carrera de un atleta aficionado para determinar cuánto le falta para superar un récord, y calcula el promedio de sus tiempos.



Cada uno de estos problemas es enviado a ChatGPT con el prompt general previamente definido. El modelo genera los respectivos códigos en Python junto con explicaciones claras sobre su funcionamiento y los recursos utilizados en cada caso.

Este contenido es ideal para docentes, estudiantes o personas interesadas en aprender a estructurar problemas de programación y explorar cómo la inteligencia artificial puede apoyar el proceso de enseñanza-aprendizaje. En el próximo video se realiza la ejecución de los programas generados dentro del editor Visual Studio Code para verificar su correcto funcionamiento.

En el siguiente video se pondrán a prueba los códigos generados con ChatGPT utilizando Visual Studio Code, con el propósito de verificar que funcionen según lo esperado.

#### **Video 5.** Ejemplo 2. Probando el código en el editor de Visual Studio Code



### [Enlace de reproducción del video](#)

#### **Síntesis del video: ejemplo 2. Probando el código en el editor Visual Studio Code**

En este video se realiza la verificación práctica de los códigos en lenguaje Python generados previamente por medio de prompts diseñados y ejecutados en ChatGPT. Para ello, se emplea el entorno de desarrollo Visual Studio Code, donde se crean archivos individuales para cada programa y se procede a su ejecución y análisis.

El proceso inicia con la creación de una carpeta de trabajo donde se organizan los archivos correspondientes, cada uno con su respectiva extensión .py, propia del lenguaje Python. Se hace una breve explicación sobre la importancia de las extensiones en programación, comparándolas con otros formatos conocidos como .docx, .jpg o .mp4.

A continuación, se prueba el funcionamiento de cada programa generado:

- **Operaciones aritméticas:** se verifica que el programa solicite la operación deseada, reciba los valores correctamente y devuelva el resultado esperado.
- **Lista ordenada:** se comprueba que los nombres ingresados se organicen correctamente en orden alfabético ascendente.
- **Venta de leche:** se ingresan datos de litros vendidos durante un periodo de 15 días, verificando que el cálculo del total y del valor económico sea correcto.

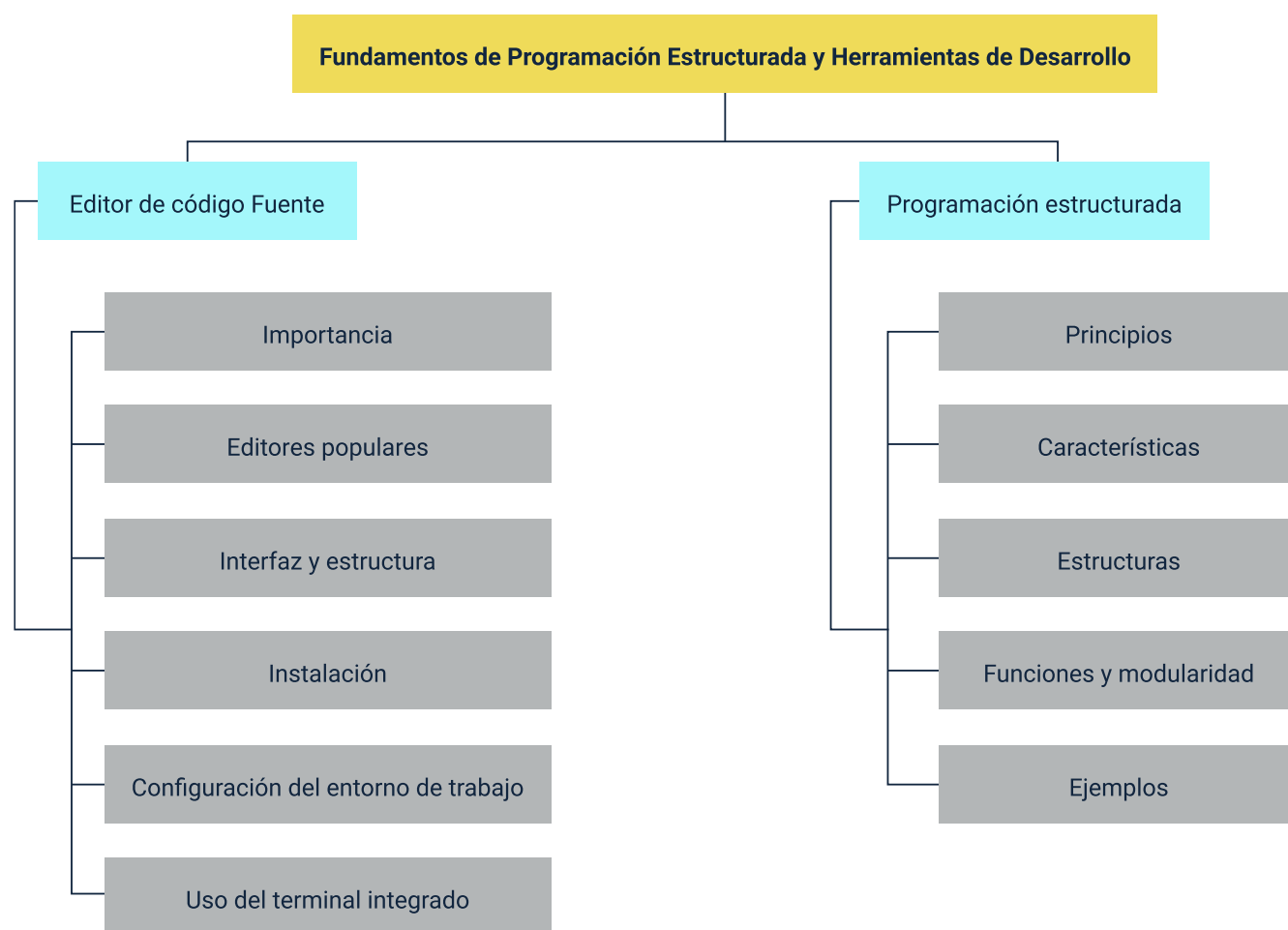
- **Entrenamiento atlético:** se ingresan los tiempos de recorrido durante cinco días y se confirma que el programa calcule la diferencia con respecto al récord histórico y el promedio de los tiempos correctamente.

Cada ejecución muestra que los programas generados por ChatGPT funcionan conforme a los requerimientos establecidos en los problemas planteados, lo que demuestra la utilidad de los prompts bien estructurados para generar soluciones automatizadas.

Este video está orientado a docentes, estudiantes de programación y cualquier persona interesada en validar código generado por IA, explorando cómo se puede integrar en procesos de enseñanza, práctica y desarrollo de software.

## 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
Editor de código fuente	Moure, M. by B. [@mouredev]. (2023). Las 15 MEJORES EXTENSIONES para VSCODE. [Archivo de video] Youtube.	Vídeo	<a href="https://www.youtube.com/watch?v=lv88bCi7eyg">https://www.youtube.com/watch?v=lv88bCi7eyg</a>
Editor de código fuente	Programador X. (2023). Visual Studio Code: Tutorial (curso completo de VSCode). [Archivo de video] Youtube.	Vídeo	<a href="https://www.youtube.com/watch?v=FzRGSihQlyY">https://www.youtube.com/watch?v=FzRGSihQlyY</a>
Editor de código fuente	Visual Studio Code - code editing. Redefined. (2025).	Sitio web	<a href="https://code.visualstudio.com/">https://code.visualstudio.com/</a>
Editor de código fuente	Geeky Script [@GeekyScript]. (2025). Cómo instalar Visual Studio Code en Windows 10/11 [Actualización 2025]. [Archivo de video] Youtube.	Vídeo	<a href="https://www.youtube.com/watch?v=mWcPAGMx8lc">https://www.youtube.com/watch?v=mWcPAGMx8lc</a>
Editor de código fuente	Roelcode [@roelcode]. (2023). Configura Visual Studio Code para Python y crear proyectos   Paso a paso. [Archivo de video] Youtube.	Vídeo	<a href="https://www.youtube.com/watch?v=MVsNOTvwwlw">https://www.youtube.com/watch?v=MVsNOTvwwlw</a>
Programación estructurada	Contreras, M. [@martincontreras9589]. (2022). Python-programación estructurada. [Archivo de video] Youtube.	Vídeo	<a href="http://www.youtube.com/playlist?list=PLcRgzfqsp7xpg1NEQ_CZMI7bLdPQIBVFm">http://www.youtube.com/playlist?list=PLcRgzfqsp7xpg1NEQ_CZMI7bLdPQIBVFm</a>

Tema	Referencia	Tipo de material	Enlace del recurso
Programación estructurada	Huet, P. (2024). Fundamentos de Python: Sintaxis, variables y estructuras. Openwebinars.net. [Archivo de video] Youtube.	Sitio web	<a href="https://openwebinars.net/blog/fundamentos-de-python-sintaxis-variables-y-estructuras-de-control/">https://openwebinars.net/blog/fundamentos-de-python-sintaxis-variables-y-estructuras-de-control/</a>
Programación estructurada	Dalto, L. [@soydalto]. (2025). Curso de PYTHON desde CERO (Completo). [Archivo de video] Youtube.	Web	<a href="https://www.youtube.com/watch?v=nKPbfIU442g">https://www.youtube.com/watch?v=nKPbfIU442g</a>

## Glosario

**Código fuente:** es el conjunto de instrucciones escritas por un programador en un lenguaje de programación legible por humanos. Representa la base de cualquier programa informático y debe ser interpretado o compilado para ejecutarse en una máquina. Su calidad influye directamente en el rendimiento, mantenimiento y seguridad del software.

**Desarrollo de software:** es el proceso de diseñar, construir, probar y mantener programas informáticos. Involucra diferentes etapas como análisis, codificación, pruebas y documentación, y puede llevarse a cabo de manera individual o colaborativa. Es una actividad clave en la transformación digital y la automatización de procesos.

**Editor de código fuente:** es una herramienta digital utilizada para escribir, editar y gestionar el código fuente de programas. Ofrece funciones como resaltado de sintaxis, autocompletado, terminal integrada y soporte para múltiples lenguajes. Facilita la productividad y organización del trabajo de los desarrolladores.

**Estructuras de control:** son construcciones que permiten controlar el flujo de un programa mediante secuencia, selección e iteración. Facilitan la toma de decisiones, la repetición de acciones y la ejecución ordenada del código.

**Extensiones:** son complementos que amplían las funcionalidades de un editor de código fuente. Permiten integrar nuevas herramientas como depuradores, asistentes de inteligencia artificial, soporte para nuevos lenguajes o gestores de versiones. Su uso personaliza y optimiza el entorno de desarrollo.

**Funciones:** son bloques de código reutilizables que realizan tareas específicas. Pueden recibir datos (parámetros) y devolver resultados, facilitando la modularización y la organización del programa.

**Interfaz:** es el conjunto de elementos visuales y de interacción que presenta una herramienta o software al usuario. En los editores de código, incluye menús, barras laterales, terminales y paneles que permiten navegar, configurar y escribir código de forma intuitiva.

**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.

**Modularidad:** es un principio que divide el programa en componentes independientes. Mejora la legibilidad, facilita el mantenimiento y permite la reutilización del código.

**Multiplataforma:** hace referencia a la capacidad de un software para funcionar en diferentes sistemas operativos como Windows, macOS y Linux sin necesidad de modificaciones. Esta característica garantiza mayor compatibilidad, accesibilidad y flexibilidad para los usuarios.

**Programación estructurada:** es un paradigma que organiza el código utilizando estructuras de control básicas y funciones. Mejora la claridad, la modularidad y la mantenibilidad del software.

**Sintaxis:** es el conjunto de reglas que define la estructura correcta de las instrucciones en un lenguaje de programación. Una sintaxis incorrecta genera errores



de compilación o ejecución. Los editores de código suelen ayudar resaltando errores de sintaxis en tiempo real.

**Terminal:** es una interfaz de línea de comandos que permite ejecutar instrucciones directamente al sistema operativo o entorno de desarrollo. En editores como Visual Studio Code, la terminal integrada facilita la ejecución de scripts, comandos Git o compilaciones sin salir del entorno de trabajo.

**Visual Studio Code:** es un editor de código fuente desarrollado por Microsoft, ampliamente utilizado por su velocidad, versatilidad y soporte para múltiples lenguajes. Incluye terminal integrada, extensiones, depuración, control de versiones y herramientas colaborativas. Es gratuito, multiplataforma y de código abierto.

## Referencias bibliográficas

- Casero, A. (2023). ¿Qué es la modularidad en programación? KeepCoding.io.  
<https://keepcoding.io/blog/que-es-la-modularidad-en-programacion/>
- Id digital school. (2024). ¿Qué es un editor de código? Id digital school.  
<https://iddigitalschool.com/bootcamps/que-es-un-editor-de-codigo/>
- Maldonado, R. (2024). Descubre la terminal de Visual Studio Code.  
KeepCoding. <https://keepcoding.io/blog/terminal-de-visual-studio-code/>
- Microsoft. (2023). Funciones (C++). Microsoft.com.  
<https://learn.microsoft.com/es-es/cpp/cpp/functions-cpp?view=msvc-170>
- Stack overflow. (2024). Integrated development environment.  
Stackoverflow.Co. <https://survey.stackoverflow.co/2024/technology#1-integrated-development-environment>
- UNIR. (2022). ¿Qué es la programación estructurada? Unir.net.  
<https://unirfp.unir.net/revista/ingenieria-y-tecnologia/programacion-estructurada/>
- UNIR. (2023). 5 editores de código indispensables para todo desarrollador.  
Unir.net. <https://www.unir.net/revista/ingenieria/editores-codigo/>
- Weisheim, R. (2022). Los 17 mejores editores de código. Hostinger; Tutoriales  
Hostinger. <https://www.hostinger.com/es/tutoriales/editores-de-codigo>

## 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 Pecuário - Regional Huila
Blanca Flor Tinoco Torres	Diseñador de contenidos digitales	Centro Agroempresarial y Desarrollo Pecuário - Regional Huila
Leyson Fabián Castaño Pérez	Desarrollador full stack	Centro Agroempresarial y Desarrollo Pecuário - Regional Huila
Carlos Eduardo Garavito Parada	Animador y productor multimedia	Centro Agroempresarial y Desarrollo Pecuário - Regional Huila
Jose Eduardo Solano Rivero	Animador y productor multimedia	Centro Agroempresarial y Desarrollo Pecuário - Regional Huila
Andrés Felipe Guevara Ariza	Locución	Centro Agroempresarial y Desarrollo Pecuário - Regional Huila
Aixa Natalia Sendoya Fernández	Validador de recursos educativos digitales	Centro Agroempresarial y Desarrollo Pecuário - Regional Huila
Jaime Hernán Tejada Llano	Validador de recursos educativos digitales	Centro Agroempresarial y Desarrollo Pecuário - Regional Huila
Raúl Mosquera Serrano	Evaluador para contenidos inclusivos y accesibles	Centro Agroempresarial y Desarrollo Pecuário - Regional Huila

Nombre	Cargo	Centro de Formación y Regional
Daniel Ricardo Mutis Gómez	Evaluador para contenidos inclusivos y accesibles	Centro Agroempresarial y Desarrollo Pecuario - Regional Huila