****

**FACULTAD DE ARQUITECTURA E INGENIERÍA**

**ESCUELA PROFESIONAL DE INGENIERÍA DE SOFTWARE**

**CURSO**

**FUNDAMENTOS DE PROGRAMACIÓN**

**PROYECTO**

Sistema de gestión financiera personal

**Integrantes:**

1. Castro Palomino, María de las Nieves
2. Espinoza Gobea, Oscar Martin
3. Hoces Azañero, Reynaldo Emilio
4. Montalvo Garcia, Antony Abel
5. Quispe Vera, Rudi Nelson
6. Vicente Puicon Jean Pierre

**Ciclo I**

**2025**

**Índice general**

[Introducción 3](#_Toc216997007)

[Marco teórico 4](#_Toc216997008)

[2.1. Definición de términos básicos 8](#_Toc216997009)

[Desarrollo 10](#_Toc216997010)

[3.1. Estructura general del programa: Entrada, Proceso y Salida 10](#_Toc216997011)

[3.2. Proceso de implementación 13](#_Toc216997012)

[3.3. Estructuras de control de programación 14](#_Toc216997013)

[3.4. Módulos de programa: funciones, procedimientos y uso de librerías 16](#_Toc216997014)

[3.5. Uso de la recursividad en el programa 18](#_Toc216997015)

[Metodología o enfoque 20](#_Toc216997016)

[4.1. Métodos utilizados para recopilar y analizar la información 20](#_Toc216997017)

[4.2. Metodología ágil utilizada en el proyecto 21](#_Toc216997018)

[4.3. Cronograma de actividades 22](#_Toc216997019)

[4.4. Actas de la reunión 22](#_Toc216997020)

[Conclusiones 22](#_Toc216997021)

[Recomendaciones 23](#_Toc216997022)

[Referencias 25](#_Toc216997023)

[Anexos 27](#_Toc216997024)

[Pseudocódigo implementado con PSeInt 27](#_Toc216997025)

[Evidencia de ejecución del programa 30](#_Toc216997026)

[Código en Python (Notebook) 32](#_Toc216997027)

# Introducción

En el marco del curso de Fundamentos de Programación, desarrollamos el proyecto denominado *Sistema de Gestión Financiera Personal*, cuyo propósito es aplicar los principios básicos de la programación estructurada en la resolución de un problema real y cercano al contexto del estudiante: la organización y administración de las finanzas personales. El curso tiene como objetivo principal que podamos adquirir competencias iniciales en el diseño de algoritmos, el uso del modelo Entrada–Proceso–Salida y la correcta estructuración de programas mediante secuencia, selección y repetición, promoviendo soluciones claras, legibles y correctamente organizadas, tal como se establece en los enfoques clásicos de la ingeniería de software [1] , [2].

El presente proyecto se alinea con dichos objetivos académicos al integrar conceptos fundamentales de programación con nociones elementales de gestión financiera personal. Desde la perspectiva del desarrollo de software, se busca que el estudiante comprenda la importancia de estructurar un programa de manera ordenada, modular y comprensible, incluso en sistemas sencillos, siguiendo buenas prácticas de codificación y claridad lógica recomendadas por diversos autores en el ámbito de la ingeniería de software [3], [4]. En este sentido, el proyecto no solo refuerza habilidades técnicas, sino que también fomenta el análisis y la planificación, competencias esenciales en la formación de un ingeniero.

El objetivo del proyecto dentro del curso es diseñar e implementar un algoritmo que permita al usuario registrar sus ingresos mensuales (diferenciando entre ingresos fijos y variables) así como clasificar sus gastos en categorías representativas de la vida cotidiana, tales como gastos necesarios, gastos discrecionales y ahorro. A partir de esta información, el sistema calcula la distribución porcentual del ingreso y ofrece recomendaciones orientativas basadas en criterios ampliamente aceptados en la educación financiera, como la regla del ahorro 50–30–20, difundida por entidades financieras y especialistas en finanzas personales [5], [6].

La problemática abordada se origina en la dificultad que enfrentan muchas personas para identificar con claridad en qué categorías concentran sus gastos y si su distribución financiera es adecuada. La ausencia de una clasificación sistemática de ingresos y egresos suele generar desequilibrios económicos y limita la capacidad de ahorro a mediano y largo plazo. Frente a esta situación, el sistema propuesto plantea una solución computacional básica pero significativa, que demuestra cómo, mediante el uso de algoritmos sencillos y estructuras de control elementales, es posible apoyar la toma de decisiones financieras informadas. Asimismo, el proyecto evidencia cómo los principios del diseño de software, incluso en aplicaciones de pequeña escala, contribuyen a la construcción de soluciones comprensibles, reutilizables y alineadas con buenas prácticas de desarrollo.

# Marco teórico

El desarrollo del sistema de gestión financiera personal se fundamenta en principios básicos de programación, conceptos introductorios de ingeniería de software y modelos reconocidos de planificación financiera personal. Estos enfoques permiten abordar el problema desde una perspectiva técnica y formativa, alineada con los objetivos del curso y con la necesidad de resolver problemas reales mediante algoritmos claros y estructurados.

**A. Programación y desarrollo de software**

La programación constituye el proceso mediante el cual se diseñan soluciones computacionales a problemas específicos, a través de la definición ordenada de instrucciones. Pressman y Maxim señalan que el desarrollo de software debe partir de una comprensión clara del problema y avanzar progresivamente hacia soluciones simples, verificables y bien estructuradas, especialmente en contextos formativos [1]. En este sentido, el proyecto prioriza la construcción de un programa sencillo, pero funcional, que represente adecuadamente un problema cotidiano como la gestión de las finanzas personales.

Asimismo, Sommerville destaca que incluso los sistemas pequeños deben diseñarse con criterios de claridad y organización, ya que esto facilita su comprensión, validación y mantenimiento [2]. En el contexto académico, este enfoque resulta esencial, pues el aprendizaje del proceso de desarrollo es tan importante como el resultado obtenido.

**B. Programación estructurada**

La programación estructurada es un paradigma que propone la organización del código mediante el uso exclusivo de tres estructuras de control: secuencia, selección y repetición. Este enfoque reduce la complejidad del programa y minimiza errores lógicos, favoreciendo un flujo de ejecución claro y predecible [3].

McConnell enfatiza que la correcta estructuración del código mejora significativamente su legibilidad y facilita su depuración, aspectos fundamentales en entornos educativos donde el código debe ser comprendido y evaluado por terceros [3]. En el sistema desarrollado, este paradigma se aplica para organizar de manera lógica el ingreso de datos financieros, su procesamiento y la presentación de resultados al usuario.

**C. Modelo Entrada–Proceso–Salida (EPS)**

El modelo Entrada–Proceso–Salida (EPS) constituye una herramienta conceptual clave para el análisis y diseño de algoritmos. Este modelo establece que todo sistema computacional recibe datos de entrada, los transforma mediante un proceso y genera información de salida. Sommerville resalta que esta separación permite un diseño más organizado y facilita la identificación de errores durante el desarrollo del software [2].

En el presente proyecto, el modelo EPS guía la estructura general del programa: los datos ingresados por el usuario, como ingresos y gastos mensuales, conforman la entrada; los cálculos de totales, clasificaciones y porcentajes corresponden al proceso; y los reportes generados representan la salida. Esta organización refuerza la comprensión del funcionamiento interno del sistema y promueve buenas prácticas de programación.

**D. Calidad y legibilidad del código**

La calidad del código es un aspecto clave en el desarrollo de software. Martin sostiene que un código limpio debe ser fácil de leer y entender, ya que la legibilidad es una condición necesaria para su mantenimiento y reutilización [4]. En el ámbito académico, esta idea se traduce en el uso de nombres descriptivos, una correcta indentación y la inclusión de comentarios explicativos.

Pressman y Maxim complementan esta idea señalando que la documentación interna del código facilita la evaluación, comprensión y replicación del programa, aspectos particularmente relevantes en proyectos formativos [1].

En consecuencia, el sistema propuesto incorpora comentarios que describen la lógica y finalidad de cada bloque del algoritmo.

**E. Gestión financiera personal y modelo 50–30–20**

La gestión financiera personal se refiere al proceso de planificación, control y evaluación de los ingresos, gastos y ahorros de una persona. Una de las principales dificultades en este ámbito es la falta de clasificación y seguimiento de los gastos, lo que impide una adecuada toma de decisiones económicas [5], [6].

Un modelo ampliamente reconocido para la organización de las finanzas personales es la regla 50–30–20, propuesta y difundida por Elizabeth Warren y Amelia Warren Tyagi en su obra *All Your Worth: The Ultimate Lifetime Money Plan* [8]. Este modelo establece que los ingresos mensuales deben distribuirse en tres categorías principales: 50 % para gastos necesarios, 30 % para gastos discrecionales y 20 % para el ahorro. Su principal fortaleza radica en ofrecer una guía simple y flexible que facilita el control financiero sin requerir conocimientos avanzados de economía.

Diversas entidades financieras, como BBVA y Scotiabank, han adoptado y difundido este modelo como una estrategia práctica para mejorar la educación financiera y promover hábitos de ahorro responsables [5], [6]. En el sistema de gestión financiera personal desarrollado, esta regla se utiliza como referencia conceptual para clasificar los gastos ingresados por el usuario y generar recomendaciones orientativas sobre su comportamiento financiero.

**F. Pensamiento algorítmico y abstracción**

El diseño del sistema también se apoya en el pensamiento algorítmico, entendido como la capacidad de descomponer un problema en pasos finitos y ordenados. Knuth señala que los algoritmos constituyen la base de toda solución computacional y que su correcta formulación es esencial para garantizar la eficiencia y corrección del programa [8].

En este proyecto, el pensamiento algorítmico se aplica para abstraer la realidad financiera del usuario y representarla mediante un conjunto de instrucciones claras, comprensibles y ejecutables en un entorno de pseudocódigo como PSeInt.

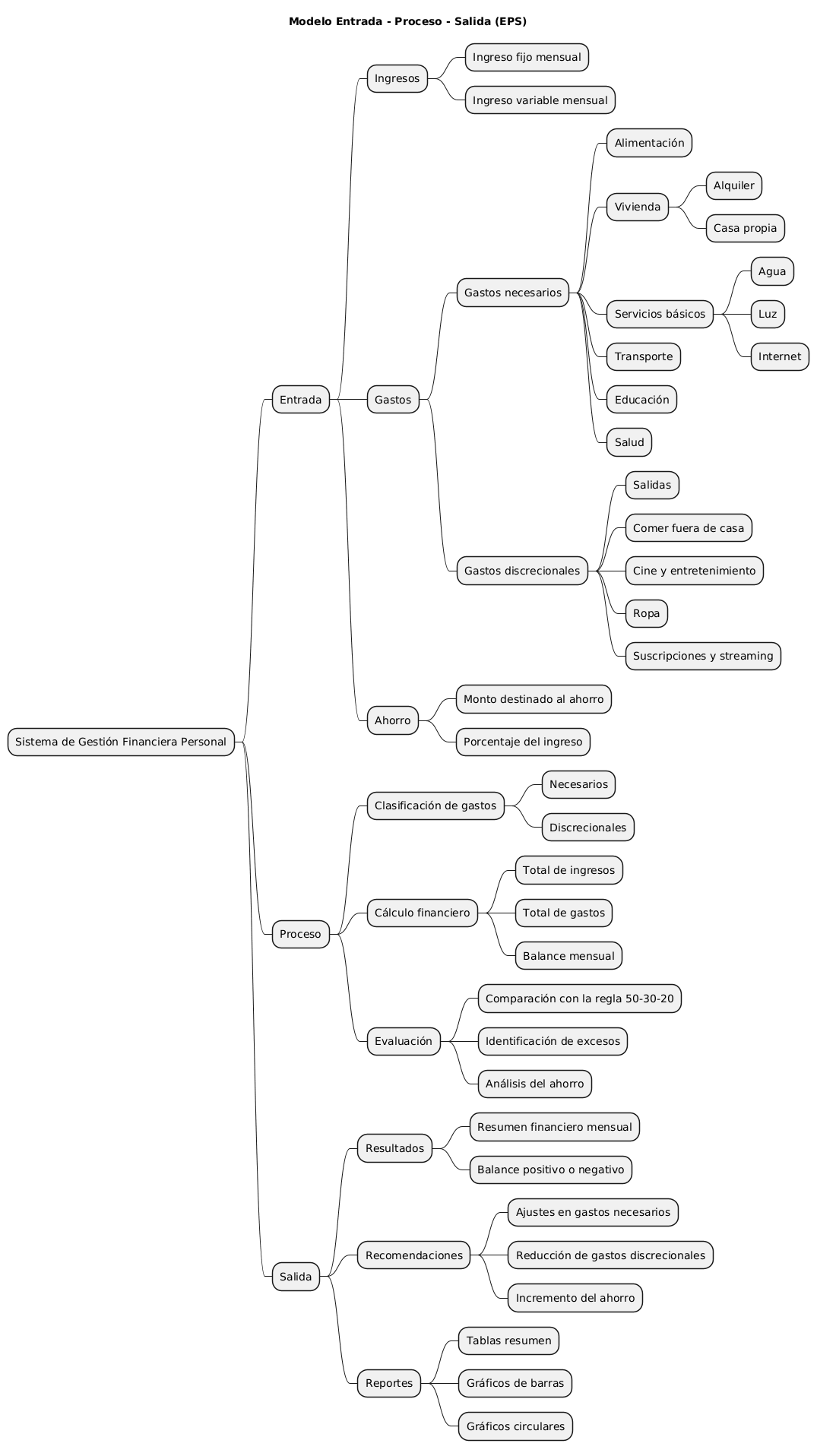


Figura 1. Mapa conceptual del modelo E–P–S.

## 2.1. Definición de términos básicos

* **Algoritmo:** Es un conjunto ordenado y finito de pasos que permiten resolver un problema de manera sistemática. Representa la lógica fundamental que guía la ejecución de cualquier programa informático [9].
* **Constante:** Se refiere a un valor fijo que permanece inalterable y no cambia durante todo el tiempo de ejecución del programa [10].
* **Variable:** Espacio en memoria física destinado a almacenar un valor que tiene la capacidad de modificarse a lo largo de la ejecución de un programa [11].
* **Proceso:** Es el conjunto articulado de operaciones lógicas y aritméticas que transforman los datos de entrada para generar resultados o salidas específicas [9].
* **Entrada:** Son los datos suministrados por el usuario o capturados del entorno que resultan indispensables para iniciar el proceso de un algoritmo [9].
* **Salida:** Representa los resultados finales generados tras el procesamiento de los datos, los cuales son presentados al usuario a través de la pantalla u otros medios periféricos [9].
* **Tipo de dato:** Es una clasificación técnica que indica el tipo de valor que una variable puede almacenar, definiendo las operaciones que se pueden realizar con él (ejemplos: numérico, cadena de texto, lógico/booleano, etc.) [11].
* **Ejecución Secuencial:** Es el modo de operación más elemental en programación, donde las instrucciones se ejecutan una tras otra, en el orden exacto en que aparecen en el código fuente, de arriba hacia abajo. En este flujo, cada instrucción debe finalizar completamente antes de que comience la siguiente, asegurando que el estado del programa evolucione de manera predecible [9].
* **Estructuras Condicionales (if, else, elif):** Son herramientas de control de flujo que permiten ejecutar diferentes bloques de código en función de si se cumple o no una condición determinada. El if evalúa una expresión inicial, el elif permite evaluar condiciones adicionales si la anterior fue falsa, y el else define un bloque de respaldo si ninguna se cumple [11].
* **Bucles (for, while):** Son instrucciones que permiten ejecutar un bloque de código varias veces de manera automática. El bucle while se repite mientras una condición sea verdadera, y el bucle for recorre un elemento iterable como una lista o un rango de números [11].
* **Módulo (General):** Es un archivo con extensión .py que funciona como un contenedor o espacio de nombres para albergar funciones, variables o clases reutilizables. Facilita la organización del código y evita la duplicación de lógica [10].
* **Módulo de Funciones:** Es un archivo diseñado para agrupar definiciones de funciones relacionadas para ser usadas en diferentes programas sin copiar el código. Permite separar la lógica del programa de la interfaz de usuario [11].
* **Módulo de Clases:** Es una unidad de software que agrupa definiciones de objetos (clases) con características y comportamientos comunes, permitiendo la encapsulación. Facilita la reutilización mediante la herencia [12].
* **Módulo de Variables:** Es un archivo .py diseñado para almacenar y organizar datos constantes, configuraciones globales o estados compartidos accesibles desde distintas partes de un programa. Centraliza información como rutas o parámetros para facilitar el control desde un solo punto[10]**.**
* **Recursividad:** Es una técnica de programación donde una función se llama a sí misma para resolver un problema mediante la estrategia de "divide y vencerás". Debe incluir un caso base y un caso recursivo para evitar ejecuciones infinitas[9]**.**
* **Caso Base:** Es la condición de parada fundamental en una función recursiva que permite finalizar las llamadas a sí misma y comenzar el retorno de valores. Representa el escenario más simple cuya solución es inmediata[9]**.**
* **Caso Recursivo:** Es la parte de la función donde el problema se divide en una versión más pequeña de sí mismo mediante una nueva llamada a la propia función. Su fin es aproximar el estado del programa hacia el caso base [9].
* **Análisis de la Complejidad en Algoritmos Recursivos:** Determina el consumo de tiempo y memoria en función del tamaño de la entrada (n). En la complejidad logarítmica , el algoritmo reduce el problema en cada paso mediante una división, evaluando el crecimiento de llamadas respecto a la pila de ejecución [9].

# Desarrollo

## 3.1. Estructura general del programa: Entrada, Proceso y Salida

La estructura general del programa desarrollado se fundamenta en el modelo **Entrada–Proceso–Salida (EPS),** el cual permite organizar de manera clara y lógica el flujo de información dentro del sistema de gestión financiera personal.

Este enfoque facilita la comprensión del algoritmo, promueve una correcta separación de responsabilidades y contribuye a la legibilidad del código, aspectos esenciales tanto en el aprendizaje de la programación como en la evaluación académica del proyecto.

**A. Entrada**

La etapa de entrada corresponde a la recopilación de la información proporcionada por el usuario. En esta fase, el programa solicita los datos necesarios para describir la situación financiera mensual, considerando tanto los ingresos como los gastos y el ahorro. Los ingresos se registran diferenciando entre ingresos fijos y variables, mientras que los gastos se ingresan de manera clasificada en categorías representativas de la vida cotidiana, tales como gastos necesarios y gastos discrecionales. Asimismo, se consulta al usuario sobre el monto o porcentaje destinado al ahorro.

El diseño de esta etapa prioriza la claridad de las instrucciones y la validación básica de los datos ingresados, con el objetivo de evitar errores que puedan afectar el procesamiento posterior. De esta manera, se garantiza que la información de entrada sea coherente y adecuada para su análisis.

**B. Proceso**

La etapa de proceso constituye el núcleo lógico del programa. En esta fase, los datos ingresados son organizados, clasificados y utilizados para realizar los cálculos correspondientes. El sistema determina el total de ingresos, el total de gastos necesarios, el total de gastos discrecionales y el monto destinado al ahorro, permitiendo obtener un balance financiero mensual.

Adicionalmente, el programa evalúa la distribución de los ingresos en función de la regla del ahorro 50–30–20, comparando los porcentajes calculados con los valores de referencia. Este análisis permite identificar posibles desequilibrios, como un exceso en gastos discrecionales o un nivel de ahorro insuficiente. La lógica del proceso se implementa mediante estructuras de control simples y ordenadas, siguiendo los principios de la programación estructurada y favoreciendo la comprensión del algoritmo.

**C. Salida**

La etapa de salida se encarga de presentar los resultados del análisis financiero al usuario de manera clara y comprensible. El programa muestra un resumen de los ingresos, gastos y ahorro, así como el balance mensual obtenido. Además, se generan mensajes orientativos que informan si la distribución financiera se encuentra alineada con el modelo de referencia o si se recomienda realizar ajustes en alguna categoría.

Esta fase cumple una función informativa y educativa, ya que permite al usuario reflexionar sobre sus hábitos financieros y comprender el impacto de sus decisiones económicas. La presentación de los resultados se realiza mediante mensajes descriptivos y ordenados, evitando sobrecargar al usuario con información innecesaria.

**D. Importancia de la organización y legibilidad del código**

La correcta organización del programa y la legibilidad del código son aspectos prioritarios en el desarrollo del sistema. Para ello, el algoritmo se estructura en bloques claramente identificados correspondientes a la entrada, el proceso y la salida, utilizando nombres descriptivos para las variables y comentarios que explican la lógica de cada sección. Esta práctica facilita la comprensión del funcionamiento del programa, su evaluación académica y su posible replicación por otros estudiantes.

Asimismo, una estructura clara y bien documentada permite detectar errores con mayor facilidad y refuerza el aprendizaje de buenas prácticas de programación desde etapas iniciales. En este sentido, la aplicación del modelo EPS no solo contribuye al correcto funcionamiento del sistema, sino que también cumple un rol pedagógico fundamental en la formación del estudiante.

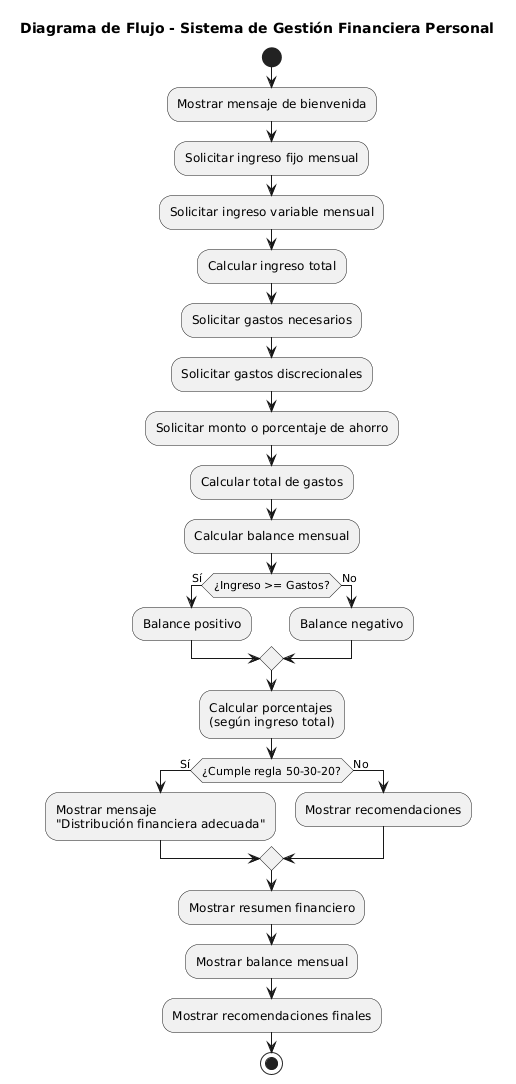


Figura 2. Diagrama de flujo del sistema.

La organización del código sigue una estructura plana (compatible con PSeInt y las restricciones del curso): bloques comentados para identificar las secciones de entrada, proceso y salida; variables declaradas al inicio con nombres significativos (ingreso, gasto, saldo, estado); y uso de una única rutina principal que contiene el menú y el ciclo de repetición. Esta estructura facilita la lectura lineal del algoritmo y su evaluación pedagógica.

## 3.2. Proceso de implementación

El proceso de implementación del *Sistema de Gestión Financiera Personal* se llevó a cabo de manera progresiva, partiendo del análisis del problema y la definición de los requisitos funcionales básicos, hasta la elaboración del algoritmo final en pseudocódigo. En primer lugar, se identificaron las variables principales del sistema, tales como los ingresos, los gastos clasificados por categorías y el ahorro, las cuales constituyen la base de la información financiera a procesar.

Posteriormente, se diseñó la estructura general del programa siguiendo el modelo Entrada–Proceso–Salida (EPS), lo que permitió organizar de forma clara el flujo del algoritmo. En esta etapa, se definió la secuencia lógica para el ingreso de datos por parte del usuario, el procesamiento de la información mediante cálculos aritméticos simples y la presentación de los resultados finales. Esta organización facilitó la traducción del diseño conceptual al pseudocódigo, asegurando coherencia entre el planteamiento teórico y la implementación práctica.

Durante la implementación en PSeInt, se optó por el uso de estructuras de control básicas, como decisiones condicionales y secuencias ordenadas, evitando el empleo de funciones avanzadas o declaraciones complejas. Esta decisión respondió a la necesidad de garantizar la compatibilidad del algoritmo con distintas versiones del entorno PSeInt utilizadas por los estudiantes, así como a mantener el enfoque pedagógico del curso centrado en los fundamentos de la programación estructurada.

Asimismo, se realizaron ajustes relacionados con la forma de ingreso de los datos financieros. En lugar de solicitar valores generales, se decidió desglosar los gastos en categorías específicas, tales como gastos necesarios y gastos discrecionales, con el objetivo de reflejar de manera más realista la situación financiera del usuario. Este ajuste permitió mejorar el análisis del proceso y facilitar la evaluación de la distribución del ingreso según la regla 50–30–20.

Otro ajuste relevante consistió en la incorporación de comentarios explicativos dentro del pseudocódigo, describiendo la lógica de cada bloque del programa. Esta práctica contribuye a mejorar la legibilidad del algoritmo, facilita su comprensión durante la evaluación académica y permite que el programa pueda ser replicado o modificado con mayor facilidad. Además, se implementaron validaciones básicas para asegurar que los valores ingresados sean coherentes, evitando resultados incorrectos o inconsistentes.

En conjunto, el proceso de implementación y los ajustes realizados permitieron consolidar un sistema funcional, claro y acorde con los objetivos del curso. El programa resultante demuestra que, mediante el uso de estructuras simples y una correcta organización del código, es posible desarrollar una solución computacional que aborde una problemática real, reforzando tanto las competencias técnicas en programación como la comprensión de conceptos básicos de gestión financiera personal.

## 3.3. Estructuras de control de programación

En el desarrollo del *Sistema de Gestión Financiera Personal* utilizamos estructuras de control básicas del lenguaje de programación Python, seleccionadas de acuerdo con los requerimientos del proyecto y el nivel del curso. Estas estructuras permiten definir el flujo de ejecución del programa, realizar evaluaciones sobre la información ingresada y controlar la repetición de determinadas operaciones, garantizando un comportamiento lógico y comprensible del sistema.

**A. Ejecución secuencial**

La ejecución secuencial constituye la base del funcionamiento del programa. En Python, las instrucciones se ejecutan en el orden en que son escritas, lo cual resulta adecuado para estructurar las etapas iniciales y finales del sistema. En el proyecto, la ejecución secuencial se emplea para:

* Mostrar mensajes informativos al usuario.
* Solicitar el ingreso de los datos financieros mensuales.
* Asignar los valores ingresados a las variables correspondientes.
* Calcular los totales de ingresos, gastos y ahorro.
* Mostrar el resumen financiero final.

Esta estructura permite que el programa siga un flujo lógico claro, facilitando la lectura del código y la comprensión del funcionamiento general del sistema.

**B. Estructuras condicionales (if, elif, else)**

Las estructuras condicionales en Python, implementadas mediante las sentencias if, elif y else, se utilizan para tomar decisiones en función de los valores ingresados y los resultados obtenidos durante el procesamiento de la información. En el sistema desarrollado, estas estructuras se emplean específicamente para:

* Evaluar si el balance mensual es positivo, negativo o nulo.
* Comparar la distribución porcentual de los gastos necesarios, gastos discrecionales y ahorro con los valores de referencia de la regla 50–30–20.
* Generar mensajes y recomendaciones distintas según la situación financiera del usuario.
* Validar que los valores ingresados sean mayores o iguales a cero.

El uso de condicionales permite que el sistema responda de manera dinámica a diferentes escenarios financieros, haciendo que el programa sea más interactivo y útil desde el punto de vista formativo.

**C. Estructuras repetitivas (while y for)**

Las estructuras repetitivas se utilizan para ejecutar un conjunto de instrucciones varias veces. En el proyecto, se emplean de manera controlada y con propósitos bien definidos:

* El bucle while se utiliza para:
  + Controlar la ejecución del programa mientras el usuario desee continuar utilizando el sistema.
  + Repetir el ingreso de datos cuando se detecta un valor inválido.
  + Gestionar un menú de opciones, en caso de que el sistema permita múltiples registros.
* El bucle for se utiliza para:
  + Recorrer un conjunto de categorías de gastos previamente definidas.
  + Sumar los montos correspondientes a cada tipo de gasto.
  + Procesar listas simples de valores financieros cuando sea necesario.

Estas estructuras permiten reducir la repetición innecesaria de código, mejorar la organización del programa y hacer que el sistema sea más flexible ante distintas situaciones de uso.

## 3.4. Módulos de programa: funciones, procedimientos y uso de librerías

El Sistema de Gestión Financiera Personal se estructura de manera modular, siguiendo principios de descomposición funcional, con el propósito de mejorar la legibilidad, mantenibilidad y reutilización del código. En el contexto del lenguaje de programación Python, esta modularización se implementa principalmente mediante funciones, dado que Python no distingue formalmente entre funciones y procedimientos como otros lenguajes; sin embargo, conceptualmente se adopta dicha diferenciación para efectos didácticos y de diseño.

**A. Funciones y procedimientos**

En el desarrollo del sistema, se emplean:

* **Funciones con valor de retorno**, encargadas de realizar cálculos específicos y devolver resultados que serán utilizados en otros módulos del programa.  
  Ejemplos conceptuales:
  + Cálculo del total de ingresos mensuales.
  + Cálculo del total de gastos por categoría.
  + Evaluación del porcentaje de ahorro respecto al ingreso total.
* **Procedimientos (funciones sin retorno explícito)**, orientados a la ejecución de tareas operativas, como:
  + Solicitud y validación de datos ingresados por el usuario.
  + Presentación de resultados en pantalla.
  + Generación de reportes descriptivos.

Esta separación conceptual permite que cada bloque del programa tenga una responsabilidad clara, alineándose con buenas prácticas de programación y diseño de software.

**B. Uso de funciones predefinidas del lenguaje**

El sistema hace uso de diversas **funciones incorporadas de Python**, las cuales facilitan la implementación eficiente del programa, entre ellas:

* input() para la entrada de datos del usuario.
* print() para la salida de información estructurada.
* float() e int() para la conversión de tipos de datos.
* sum() para el cálculo acumulado de ingresos y gastos.
* len() para el control de estructuras iterativas cuando se requiera.

Estas funciones permiten una implementación clara y comprensible, especialmente adecuada para un contexto formativo.

**C. Librerías utilizadas**

Con el fin de enriquecer el sistema y permitir una futura extensión hacia la visualización de datos y generación de reportes, se contempla el uso de las siguientes librerías estándar y externas de Python:

* **datetime**: para registrar periodos mensuales y gestionar fechas asociadas a los registros financieros.
* **matplotlib** (de manera proyectiva): para la generación de gráficos de barras y circulares que representen ingresos, gastos y distribución presupuestal.
* **csv o pandas** (opcional): para el almacenamiento estructurado de los datos financieros y su posterior análisis.
* **reportlab o herramientas equivalentes** (fase proyectada): para la generación de reportes en formato PDF.

El uso de estas librerías responde a la necesidad de transformar datos numéricos en información comprensible y visual, fortaleciendo el carácter práctico del sistema.

**D. Importancia del enfoque modular**

La organización del programa en módulos independientes aporta múltiples beneficios:

* Facilita la comprensión del código por parte de estudiantes y evaluadores.
* Permite realizar ajustes o ampliaciones sin afectar el funcionamiento global del sistema.
* Favorece la detección y corrección de errores.
* Refuerza el aprendizaje de buenas prácticas de programación estructurada.

En conjunto, esta estructura modular constituye una base sólida para el desarrollo progresivo del sistema, manteniendo un equilibrio entre simplicidad, funcionalidad y escalabilidad.

## 3.5. Uso de la recursividad en el programa

La recursividad es una técnica de programación mediante la cual una función se define en términos de sí misma, es decir, una función puede invocarse nuevamente durante su propia ejecución hasta que se cumple una condición de término. Esta técnica resulta especialmente útil para resolver problemas que pueden descomponerse en subproblemas de la misma naturaleza, siempre que exista un caso base que evite la ejecución infinita.

En el contexto del presente Sistema de Gestión Financiera Personal, la recursividad no constituye el mecanismo principal de procesamiento, dado que la lógica del programa se basa principalmente en la ejecución secuencial, el uso de estructuras condicionales y bucles iterativos. Sin embargo, se incorpora de manera didáctica y controlada para reforzar los conceptos fundamentales de programación vistos en el curso y demostrar su correcta comprensión.

**A. Aplicación conceptual de la recursividad**

Dentro del programa, la recursividad puede emplearse para realizar tareas simples y repetitivas, tales como:

* Recalcular proyecciones de ahorro mes a mes.
* Simular el crecimiento acumulado del ahorro en un periodo determinado.
* Validar el ingreso de datos cuando el usuario introduce valores incorrectos.

Estas aplicaciones permiten ilustrar el funcionamiento de la recursividad sin comprometer la claridad ni la legibilidad del código principal.

**B. Ejemplo de función recursiva básica en Python**

A modo ilustrativo, se presenta una función recursiva sencilla que calcula el ahorro acumulado tras *n* meses, suponiendo que el usuario mantiene un monto fijo de ahorro mensual:

def ahorro\_acumulado(meses, ahorro\_mensual):

if meses == 0:

return 0

else:

return ahorro\_mensual + ahorro\_acumulado(meses - 1, ahorro\_mensual)

En este ejemplo:

* El caso base ocurre cuando el número de meses es igual a cero.
* La llamada recursiva reduce progresivamente el número de meses hasta alcanzar el caso base.
* El resultado final representa el ahorro total acumulado.

**C. Consideraciones sobre el uso de la recursividad**

Si bien la recursividad es una herramienta poderosa, en este proyecto se prioriza el uso de estructuras iterativas (for, while) para garantizar un mejor rendimiento y una mayor facilidad de comprensión por parte de los estudiantes. La inclusión de funciones recursivas responde principalmente a un objetivo formativo, más que a una necesidad estricta del sistema.

De este modo, el programa demuestra el conocimiento de esta estructura avanzada de control, manteniendo al mismo tiempo una implementación clara, organizada y coherente con los objetivos del curso.

# Metodología o enfoque

## 4.1. Métodos utilizados para recopilar y analizar la información

Para el desarrollo del *Sistema de Gestión Financiera Personal* se adoptó un enfoque metodológico de tipo aplicado, orientado a la solución de un problema concreto mediante la implementación de un programa informático educativo. La recopilación de información se realizó principalmente a partir de fuentes bibliográficas especializadas en ingeniería de software, buenas prácticas de programación y educación financiera personal, lo que permitió sustentar teóricamente las decisiones de diseño y desarrollo del sistema [1], [2], [3].

Desde el punto de vista financiero, se analizaron modelos reconocidos de planificación financiera personal, en particular la regla del ahorro 50–30–20, difundida por entidades financieras y respaldada por literatura especializada en gestión del dinero [5], [6], [13]. Este modelo permitió establecer criterios claros para la clasificación de ingresos, gastos necesarios, gastos discrecionales y ahorro, los cuales fueron incorporados como elementos centrales del programa.

El análisis de los datos ingresados por el usuario se realiza de forma directa y transparente, utilizando operaciones aritméticas básicas y estructuras de control del lenguaje Python. Los datos se ingresan a través de la consola, se procesan en tiempo de ejecución y se presentan como resultados comprensibles para el usuario, sin el uso de bases de datos ni herramientas externas. Este método es coherente con los objetivos del curso, que buscan fortalecer la comprensión de los fundamentos de la programación, tales como la manipulación de variables, el control del flujo del programa y la correcta estructuración del código [8].

## 4.2. Metodología ágil utilizada en el proyecto

El desarrollo del sistema se realizó siguiendo un enfoque ágil e incremental, basado en los principios propuestos por las metodologías ágiles modernas. De acuerdo con Pressman y Maxim [1] y Sommerville [2], este tipo de enfoque resulta adecuado para proyectos educativos, ya que permite adaptarse progresivamente a los requerimientos y realizar mejoras continuas a partir de versiones funcionales del software.

En particular, se tomaron como referencia los lineamientos generales de Scrum, los cuales promueven el desarrollo iterativo, la entrega temprana de funcionalidades y la revisión constante del producto [14]. En este proyecto, el sistema fue construido inicialmente con una estructura básica que permitía el ingreso y procesamiento de datos financieros, para luego incorporar mejoras relacionadas con la clasificación detallada de gastos y la generación de recomendaciones financieras.

Asimismo, se consideraron principios de Extreme Programming (XP), especialmente aquellos relacionados con la simplicidad del diseño, la claridad del código y la retroalimentación continua [15]. Estas prácticas se reflejan en la organización modular del programa, el uso de funciones bien definidas y la inclusión de comentarios explicativos que facilitan la comprensión y evaluación del código.

La adopción de un enfoque ágil permitió realizar ajustes relevantes durante el proceso de implementación, tales como la simplificación del pseudocódigo para garantizar su compatibilidad con distintas versiones de PSeInt y la priorización de estructuras de control básicas sobre construcciones más avanzadas. De este modo, la metodología empleada contribuyó a un desarrollo ordenado, flexible y alineado con los objetivos formativos del curso.

## 4.3. Cronograma de actividades

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Actividad** | **S1** | **S2** | **S3** | **S4** | **S5** | **S6** | **S7** | **S8** |
| Análisis del problema y objetivos |  | |  |  |  |  |  |  |
| Revisión bibliográfica |  | | |  |  |  |  |  |
| Diseño del sistema |  |  | |  |  |  |  |  |
| Diseño de diagramas (EPS, flujo) |  |  |  | |  |  |  |  |
| Implementación en Python |  |  |  |  | |  |  |  |
| Pruebas y ajustes |  |  |  |  |  | |  |  |
| Documentación del informe |  |  |  |  |  |  | |  |
| Revisión final y entrega |  |  |  |  |  |  |  | |

## 4.4. Actas de la reunión

# Conclusiones

En el presente informe se desarrolló un Sistema de Gestión Financiera Personal como aplicación práctica de los fundamentos de programación abordados en el curso. A lo largo del trabajo se integraron conceptos teóricos y prácticos de la ingeniería de software, permitiendo evidenciar cómo los principios de organización, modularidad y claridad del código contribuyen a la construcción de soluciones informáticas funcionales y comprensibles.

El análisis del problema permitió identificar la dificultad que presentan muchas personas para organizar sus ingresos, gastos y hábitos de ahorro. En respuesta a esta problemática, el sistema propuesto facilita la clasificación estructurada de los ingresos mensuales, así como de los gastos necesarios y discrecionales, incorporando el modelo financiero de la regla 50–30–20 como referencia para la toma de decisiones responsables. Esta integración demuestra la aplicabilidad de modelos teóricos de gestión financiera en el desarrollo de programas educativos.

Desde el punto de vista técnico, se evidenció que el uso del modelo Entrada–Proceso–Salida constituye una base sólida para el diseño de programas claros y bien estructurados. La correcta separación entre la captura de datos, su procesamiento y la presentación de resultados permitió mejorar la legibilidad del código y facilitar su mantenimiento, cumpliendo con buenas prácticas recomendadas en la literatura especializada en ingeniería de software.

Asimismo, la adopción de un enfoque ágil e incremental favoreció el desarrollo progresivo del sistema, permitiendo realizar ajustes durante la implementación sin comprometer la estabilidad del programa. Este enfoque resultó adecuado para un contexto académico, ya que promovió la retroalimentación constante, la mejora continua y la adaptación a los requerimientos del proyecto.

Finalmente, se concluye que el sistema desarrollado cumple con los objetivos planteados en el curso, al demostrar el dominio de estructuras de control, funciones, modularidad y organización del código en Python. Además, el proyecto evidencia cómo la programación puede ser utilizada como una herramienta efectiva para resolver problemas reales, sentando las bases para futuras ampliaciones, como la incorporación de visualización de datos o la generación automática de reportes financieros.

# Recomendaciones

A partir de los resultados obtenidos en el desarrollo del Sistema de Gestión Financiera Personal, se proponen las siguientes recomendaciones, orientadas tanto a la mejora del sistema como a su aplicación práctica en contextos académicos y personales:

1. **Fortalecer el hábito de registro financiero mensual**, recomendando al usuario ingresar de manera periódica y ordenada sus ingresos y gastos. Esta práctica permite obtener información confiable para el análisis financiero personal y facilita la toma de decisiones informadas sobre el uso del dinero.
2. **Aplicar de manera progresiva la regla del ahorro 50–30–20**, especialmente en usuarios que recién comienzan a organizar sus finanzas. Se sugiere adaptar inicialmente los porcentajes según la realidad económica del usuario, priorizando la sostenibilidad del hábito de ahorro antes que el cumplimiento estricto del modelo.
3. **Ampliar gradualmente la funcionalidad del sistema**, incorporando módulos de visualización de datos, como gráficos de barras y circulares, que permitan representar de forma clara la distribución de ingresos, gastos y ahorro. Esta mejora facilitaría la comprensión de la información financiera y reforzaría el carácter educativo del sistema.
4. **Implementar mecanismos de validación adicionales**, tales como alertas cuando los gastos discrecionales superen los límites recomendados o cuando el nivel de ahorro sean insuficiente. Estas validaciones contribuirían a orientar al usuario hacia una gestión financiera más responsable.
5. **Utilizar el sistema como herramienta pedagógica**, promoviendo su empleo en cursos introductorios de programación para reforzar el aprendizaje de estructuras de control, funciones y modularidad, a través de un problema real y significativo.
6. **Documentar y comentar adecuadamente el código fuente**, asegurando que futuras modificaciones o ampliaciones del sistema puedan realizarse sin dificultad. Esta práctica es fundamental para mantener la calidad del software y facilitar su evaluación académica.
7. **Explorar futuras extensiones del proyecto**, como la generación automática de reportes en formato PDF o la persistencia de datos en archivos externos, siempre considerando el nivel del curso y los objetivos formativos planteados.

# Referencias

[1] R. S. Pressman y B. R. Maxim, *Software Engineering: A Practitioner’s Approach*. New York, NY: McGraw-Hill Education, 2014.

[2] I. Sommerville, *Software Engineering*. Harlow, UK: Pearson Education, 2015.

[3] S. McConnell, *Code Complete*. Redmond, WA: Microsoft Press, 2004.

[4] R. C. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*. Upper Saddle River, NJ: Prentice Hall, 2008.

[5] BBVA, «La regla del ahorro 50–30–20». [En línea]. Disponible en: https://www.bbva.es/finanzas-vistazo/ef/cuentas/regla-ahorro-50-30-20.html

[6] Scotiabank Perú, «Cómo organizar tus finanzas personales con la regla 50–30–20». [En línea]. Disponible en: https://www.scotiabank.com.pe/blog/organizar-finanzas-personales-regla-50-30-20

[7] E. Gamma, R. Helm, R. Johnson, y J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Boston, MA: Addison-Wesley, 1994.

[8] D. E. Knuth, *The Art of Computer Programming, Vol. 1: Fundamental Algorithms*. Reading, MA: Addison-Wesley, 1997.

[9] L. Joyanes Aguilar, *Fundamentos de Programación: Algoritmos, estructura de datos y objetos*. Madrid: McGraw-Hill, 2008. [En línea]. Disponible en: https://openlibra.com/es/book/fundamentos-de-programacion

[10] El Libro de Python, «Módulos en Python». [En línea]. Disponible en: https://ellibrodepython.com/modulos-python

[11] R. González Duque, *Python para todos*. Mundo Geek, 2010. [En línea]. Disponible en: https://persoal.citius.usc.es/eva.cernadas/informaticaparacientificos/material/libros/Python%20para%20todos.pdf

[12] Universitat Oberta de Catalunya, «Introducción a la ingeniería del software», [En línea]. Disponible en: https://openaccess.uoc.edu/bitstream/10609/147761/1/Introducci%C3%B3n%20a%20la%20ingenier%C3%ADa%20del%20software.pdf

[13] E. Warren y A. W. Tyagi, *All Your Worth: The Ultimate Lifetime Money Plan*. New York, NY: Free Press, 2005.

[14] K. Schwaber y J. Sutherland, *The Scrum Guide*. Scrum.org, 2020. [En línea]. Disponible en: https://scrumguides.org

[15] K. Beck, *Extreme Programming Explained: Embrace Change*. Boston, MA: Addison-Wesley, 2000.

# Anexos

## Pseudocódigo implementado con PSeInt

Proceso Sistema\_Gestion\_Financiera\_Personal

// =========================

// DECLARACIÓN DE VARIABLES

// =========================

Definir ingreso\_fijo, ingreso\_variable, ingreso\_total Como Real

Definir gasto\_alimentacion, gasto\_vivienda, gasto\_servicios Como Real

Definir gasto\_transporte, gasto\_educacion, gasto\_salud Como Real

Definir gasto\_ocio, gasto\_ropa, gasto\_streaming, gasto\_otros Como Real

Definir gastos\_necesarios, gastos\_discrecionales Como Real

Definir ahorro\_actual Como Real

Definir porcentaje\_necesarios, porcentaje\_discrecionales, porcentaje\_ahorro Como Real

// =========================

// ENTRADA DE DATOS

// =========================

Escribir "=== SISTEMA DE GESTIÓN FINANCIERA PERSONAL ==="

Escribir "Ingrese su ingreso fijo mensual:"

Leer ingreso\_fijo

Escribir "Ingrese su ingreso variable mensual:"

Leer ingreso\_variable

ingreso\_total <- ingreso\_fijo + ingreso\_variable

Escribir ""

Escribir "=== INGRESO DE GASTOS NECESARIOS ==="

Escribir "Gasto en alimentación:"

Leer gasto\_alimentacion

Escribir "Gasto en vivienda (alquiler o mantenimiento):"

Leer gasto\_vivienda

Escribir "Gasto en servicios (agua, luz, internet):"

Leer gasto\_servicios

Escribir "Gasto en transporte:"

Leer gasto\_transporte

Escribir "Gasto en educación:"

Leer gasto\_educacion

Escribir "Gasto en salud:"

Leer gasto\_salud

gastos\_necesarios <- gasto\_alimentacion + gasto\_vivienda + gasto\_servicios + gasto\_transporte + gasto\_educacion + gasto\_salud

Escribir ""

Escribir "=== INGRESO DE GASTOS DISCRECIONALES ==="

Escribir "Gasto en ocio y entretenimiento:"

Leer gasto\_ocio

Escribir "Gasto en ropa:"

Leer gasto\_ropa

Escribir "Gasto en streaming y suscripciones:"

Leer gasto\_streaming

Escribir "Otros gastos:"

Leer gasto\_otros

gastos\_discrecionales <- gasto\_ocio + gasto\_ropa + gasto\_streaming + gasto\_otros

Escribir ""

Escribir "Ingrese el monto que destina al ahorro mensual:"

Leer ahorro\_actual

// =========================

// PROCESO

// =========================

porcentaje\_necesarios <- (gastos\_necesarios / ingreso\_total) \* 100

porcentaje\_discrecionales <- (gastos\_discrecionales / ingreso\_total) \* 100

porcentaje\_ahorro <- (ahorro\_actual / ingreso\_total) \* 100

// =========================

// SALIDA DE RESULTADOS

// =========================

Escribir ""

Escribir "=== RESUMEN FINANCIERO MENSUAL ==="

Escribir "Ingreso total: S/ ", ingreso\_total

Escribir "Gastos necesarios: S/ ", gastos\_necesarios, " (", porcentaje\_necesarios, "%)"

Escribir "Gastos discrecionales: S/ ", gastos\_discrecionales, " (", porcentaje\_discrecionales, "%)"

Escribir "Ahorro mensual: S/ ", ahorro\_actual, " (", porcentaje\_ahorro, "%)"

Escribir ""

Escribir "=== EVALUACIÓN SEGÚN REGLA 50?30?20 ==="

Si porcentaje\_necesarios <= 50 Entonces

Escribir "Gastos necesarios dentro del rango recomendado."

SiNo

Escribir "Gastos necesarios superan el 50% recomendado."

FinSi

Si porcentaje\_discrecionales <= 30 Entonces

Escribir "Gastos discrecionales dentro del rango recomendado."

SiNo

Escribir "Gastos discrecionales superan el 30% recomendado."

FinSi

Si porcentaje\_ahorro >= 20 Entonces

Escribir "Ahorro adecuado según la regla 50?30?20."

SiNo

Escribir "Se recomienda aumentar el porcentaje de ahorro."

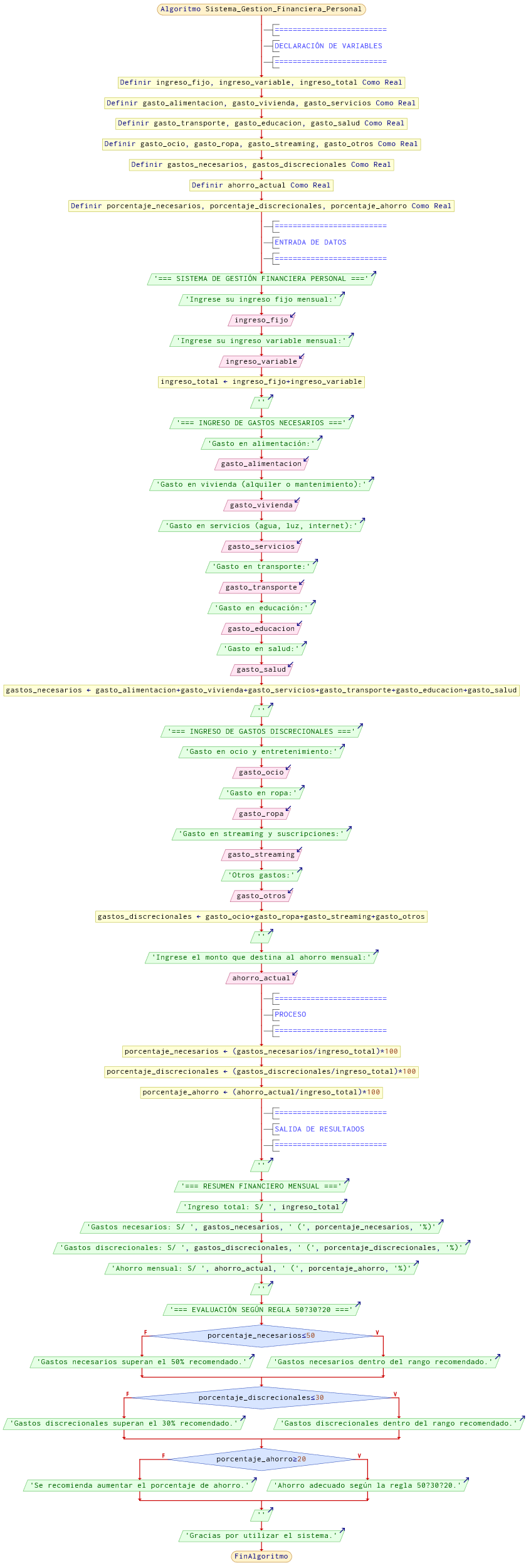
FinSi

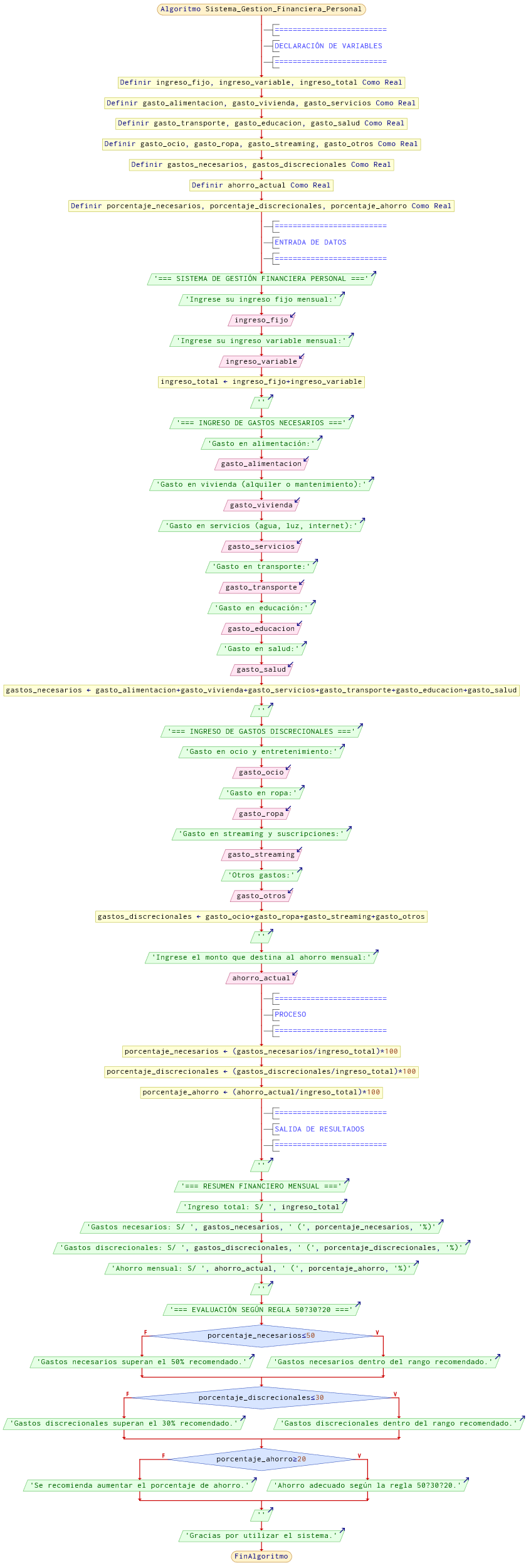
Escribir ""

Escribir "Gracias por utilizar el sistema."

FinProceso

## Evidencia de ejecución del programa





## Código en Python (Notebook)

# Sistema de gestión financiera personal

\*\*Asignatura:\*\* Fundamentos de la programación

\*\*Docente:\*\* Mo. Huanco Garcia, Analy Rosario

\*\*Integrantes:\*\*

1. Castro Palomino, María de las Nieves

2. Espinoza Gobea, Oscar Martin

3. Hoces Azañero, Reynaldo Emilio

4. Montalvo Garcia, Antony Abel

5. Quispe Vera, Rudi Nelson

6. Vicente Puicon Jean Pierre

'''

Permite crear archivos PDF desde cero mediante código Python,

sin depender de editores visuales.

'''

!pip install reportlab

# ==========================================================

# SISTEMA DE GESTIÓN FINANCIERA PERSONAL

# ==========================================================

from datetime import datetime

import matplotlib.pyplot as plt

# ==============================

# FUNCIÓN DE VALIDACIÓN NUMÉRICA

# ==============================

def leer\_numero(mensaje):

"""

Solicita un número al usuario.

No permite texto ni valores negativos.

Repite la solicitud hasta que el dato sea válido.

"""

while True:

try:

valor = float(input(mensaje))

if valor < 0:

print("El valor no puede ser negativo. Intente nuevamente.")

else:

return valor

except ValueError:

print("Entrada inválida. Por favor ingrese un número.")

# =============================

# FUNCIÓN PARA PREGUNTAS S/N

# =============================

def leer\_si\_no(mensaje):

"""

Solicita una respuesta S o N.

"""

while True:

respuesta = input(mensaje).strip().lower()

if respuesta in ["s", "n"]:

return respuesta

else:

print("Responda con 's' o 'n'.")

# Configuración de matplotlib para mejor visualización

plt.rcParams['figure.figsize'] = (10, 6)

plt.rcParams['font.size'] = 10

# =============================

# INGRESO DE DATOS

# =============================

print("=" \* 60)

print(" SISTEMA DE GESTIÓN FINANCIERA PERSONAL")

print(" Basado en la Regla 50-30-20")

print("=" \* 60)

print("\n PASO 1: INGRESOS MENSUALES")

print("-" \* 60)

ingreso\_fijo = leer\_numero("Ingreso fijo mensual (S/.): ")

ingreso\_variable = leer\_numero("Ingreso variable mensual (S/.): ")

ingreso\_total = ingreso\_fijo + ingreso\_variable

print(f"\n Ingreso total: S/. {ingreso\_total:,.2f}")

# ==========================================================

# GASTOS NECESARIOS

# ==========================================================

print("\n--- PASO 2: GASTOS NECESARIOS (50% recomendado) ---")

print("-" \* 60)

print("Incluye: vivienda, alimentación, servicios, transporte, educación, salud")

print()

print("\n--- VIVIENDA ---")

print("Si vive en casa propia, registre solo gastos de mantenimiento.")

print("Si alquila, registre el monto del alquiler.")

gasto\_vivienda = leer\_numero("Gasto en vivienda (S/.): ")

print("\n--- OTROS GASTOS NECESARIOS ---")

alimentacion = leer\_numero("Alimentación (S/.): ")

servicios = leer\_numero("Servicios (agua, luz, internet, telefonía) (S/.): ")

transporte = leer\_numero("Transporte (S/.): ")

educacion = leer\_numero("Educación (S/.): ")

salud = leer\_numero("Salud (S/.): ")

gastos\_necesarios = sum([

alimentacion, gasto\_vivienda, servicios,

transporte, educacion, salud

])

print(f"\n Total gastos necesarios: S/ {gastos\_necesarios:,.2f}")

# ==========================================================

# GASTOS DISCRECIONALES

# ==========================================================

print("\n--- PASO 3: GASTOS DISCRECIONALES (30% recomendado) ---")

print("-" \* 60)

print("Incluye: entretenimiento, ocio, ropa, suscripciones, otros")

print()

ocio = leer\_numero("Ocio y entretenimiento (S/.): ")

ropa = leer\_numero("Ropa (S/.): ")

streaming = leer\_numero("Streaming y suscripciones (S/.): ")

otros = leer\_numero("Otros gastos (S/.): ")

gastos\_discrecionales = sum([ocio, ropa, streaming, otros])

print(f"\n Total gastos discrecionales: S/ {gastos\_discrecionales:,.2f}")

# -----------------------------

# AHORRO

# -----------------------------

print("\n---PASO 4: AHORRO (20% recomendado)---")

print("-" \* 60)

respuesta\_ahorro = leer\_si\_no("\n¿Realiza ahorro mensual? (s/n): ")

if respuesta\_ahorro == "s":

ahorro = leer\_numero("Monto destinado al ahorro (S/.): ")

else:

ahorro = 0

print("Recuerde que el ahorro es importante para sus metas futuras.")

print(f"\n Ahorro mensual: S/ {ahorro:,.2f}")

# ==========================================================

# CÁLCULOS Y ANÁLISIS

# ==========================================================

total\_gastos = gastos\_necesarios + gastos\_discrecionales

balance = ingreso\_total - total\_gastos - ahorro

porc\_necesarios = (gastos\_necesarios / ingreso\_total) \* 100 if ingreso\_total > 0 else 0

porc\_discrecionales = (gastos\_discrecionales / ingreso\_total) \* 100 if ingreso\_total > 0 else 0

porc\_ahorro = (ahorro / ingreso\_total) \* 100 if ingreso\_total > 0 else 0

# Regla 50-30-20 (montos ideales)

ideal\_necesarios = ingreso\_total \* 0.50

ideal\_discrecionales = ingreso\_total \* 0.30

ideal\_ahorro = ingreso\_total \* 0.20

margen\_necesarios = ideal\_necesarios - gastos\_necesarios

margen\_discrecionales = ideal\_discrecionales - gastos\_discrecionales

margen\_ahorro = ideal\_ahorro - ahorro

# ==========================================================

# RESUMEN FINANCIERO

# ==========================================================

print("\n" + "=" \* 60)

print("RESUMEN FINANCIERO MENSUAL")

print("=" \* 60)

print(f"\n Ingreso total: S/. {ingreso\_total:>12,.2f}")

print(f" Gasto total: S/. {total\_gastos:>12,.2f}")

print(f" Ahorro: S/. {ahorro:>12,.2f}")

print(f" Balance final: S/. {balance:>12,.2f}")

if balance >= 0:

print("Balance positivo")

else:

print("Balance negativo - Revise sus gastos")

print("\n" + "-" \* 60)

print("DISTRIBUCIÓN DE GASTOS:")

print("-" \* 60)

print(f" Gastos necesarios: S/. {gastos\_necesarios:>12,.2f} ({porc\_necesarios:>5.1f}%)")

print(f" Ideal (50%): S/. {ideal\_necesarios:>12,.2f}")

print(f" Diferencia: S/. {margen\_necesarios:>12,.2f}")

print(f"\n Gastos discrecionales: S/. {gastos\_discrecionales:>12,.2f} ({porc\_discrecionales:>5.1f}%)")

print(f" Ideal (30%): S/. {ideal\_discrecionales:>12,.2f}")

print(f" Diferencia: S/. {margen\_discrecionales:>12,.2f}")

print(f"\n Ahorro: S/ {ahorro:>12,.2f} ({porc\_ahorro:>5.1f}%)")

print(f" Ideal (20%): S/. {ideal\_ahorro:>12,.2f}")

print(f" Diferencia: S/. {margen\_ahorro:>12,.2f}")

# ==========================================================

# RECOMENDACIONES PERSONALIZADAS

# ==========================================================

print("\n" + "=" \* 60)

print(" RECOMENDACIONES SEGÚN LA REGLA 50-30-20")

print("=" \* 60)

print("\n GASTOS NECESARIOS:")

if margen\_necesarios >= 0:

print(f" Sus gastos están controlados")

print(f" Puede gastar hasta S/ {margen\_necesarios:,.2f} más en necesidades")

else:

print(f" Está excediendo el límite en S/ {abs(margen\_necesarios):,.2f}")

print(f" Intente reducir gastos en: servicios, transporte o alimentación")

print("\n GASTOS DISCRECIONALES:")

if margen\_discrecionales >= 0:

print(f" Sus gastos de ocio están bajo control")

print(f" Puede gastar hasta S/ {margen\_discrecionales:,.2f} más en entretenimiento")

else:

print(f" Está excediendo el límite en S/ {abs(margen\_discrecionales):,.2f}")

print(f" Considere reducir: streaming, salidas o compras no esenciales")

print("\n AHORRO:")

if margen\_ahorro <= 0:

print(f" ¡Excelente! Está cumpliendo con la meta de ahorro")

if ahorro > ideal\_ahorro:

print(f" Está ahorrando S/ {abs(margen\_ahorro):,.2f} más de lo recomendado")

else:

print(f" Le falta ahorrar S/ {margen\_ahorro:,.2f} para cumplir la meta")

print(f" Intente incrementar su ahorro mensual gradualmente")

# ==========================================================

# GENERACIÓN DE GRÁFICOS

# ==========================================================

print("\n Generando gráficos...")

# Gráfico de barras: Distribución vs Ideal

fig, ax = plt.subplots(figsize=(12, 6))

categorias = ["Gastos\nNecesarios", "Gastos\nDiscrecionales", "Ahorro"]

valores\_reales = [gastos\_necesarios, gastos\_discrecionales, ahorro]

valores\_ideales = [ideal\_necesarios, ideal\_discrecionales, ideal\_ahorro]

x = range(len(categorias))

ancho = 0.35

barras1 = ax.bar([i - ancho/2 for i in x], valores\_reales, ancho,

label='Real', color=['#FF6B6B', '#4ECDC4', '#45B7D1'])

barras2 = ax.bar([i + ancho/2 for i in x], valores\_ideales, ancho,

label='Ideal (50-30-20)', color=['#FFA07A', '#98D8C8', '#87CEEB'], alpha=0.7)

ax.set\_ylabel('Monto (S/)', fontsize=12, fontweight='bold')

ax.set\_title('Comparación: Distribución Real vs Ideal (Regla 50-30-20)',

fontsize=14, fontweight='bold', pad=20)

ax.set\_xticks(x)

ax.set\_xticklabels(categorias, fontsize=11)

ax.legend(fontsize=11)

ax.grid(axis='y', linestyle='--', alpha=0.3)

# Agregar valores sobre las barras

for barra in barras1 + barras2:

altura = barra.get\_height()

ax.text(barra.get\_x() + barra.get\_width()/2., altura,

f'S/ {altura:,.0f}',

ha='center', va='bottom', fontsize=9)

plt.tight\_layout()

plt.savefig("barras.png", dpi=150, bbox\_inches='tight')

plt.show()

# Gráfico circular: Distribución porcentual

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6))

# Gráfico real

colores = ['#FF6B6B', '#4ECDC4', '#45B7D1']

explode = (0.05, 0.05, 0.05)

if sum([gastos\_necesarios, gastos\_discrecionales, ahorro]) > 0:

ax1.pie([gastos\_necesarios, gastos\_discrecionales, ahorro],

labels=["Necesarios", "Discrecionales", "Ahorro"],

autopct='%1.1f%%',

startangle=90,

colors=colores,

explode=explode,

shadow=True)

ax1.set\_title('Distribución Real\nde tu Ingreso', fontsize=12, fontweight='bold')

# Gráfico ideal

ax2.pie([50, 30, 20],

labels=["Necesarios\n(50%)", "Discrecionales\n(30%)", "Ahorro\n(20%)"],

autopct='%1.0f%%',

startangle=90,

colors=colores,

explode=explode,

shadow=True)

ax2.set\_title('Distribución Ideal\n(Regla 50-30-20)', fontsize=12, fontweight='bold')

plt.tight\_layout()

plt.savefig("circular.png", dpi=150, bbox\_inches='tight')

plt.show()

print("Gráficos generados exitosamente")

# ==========================================================

# GENERADOR DE REPORTE PDF

# ==========================================================

from reportlab.platypus import SimpleDocTemplate, Paragraph, Spacer, Table, Image

from reportlab.lib.styles import getSampleStyleSheet

from reportlab.lib.pagesizes import A4

from reportlab.lib.units import cm

def generar\_reporte\_pdf(datos):

"""

Genera un reporte PDF profesional con tablas,

gráficos y recomendaciones financieras.

"""

doc = SimpleDocTemplate(

"Reporte\_Financiero\_Personal.pdf",

pagesize=A4,

rightMargin=2\*cm,

leftMargin=2\*cm,

topMargin=2\*cm,

bottomMargin=2\*cm

)

estilos = getSampleStyleSheet()

contenido = []

# Título

contenido.append(Paragraph(

"Reporte de Gestión Financiera Personal",

estilos["Title"]

))

contenido.append(Spacer(1, 0.3 \* cm))

fecha = datetime.now().strftime("%d/%m/%Y %H:%M")

contenido.append(Paragraph(

f"<b>Fecha de elaboración:</b> {fecha}",

estilos["Normal"]

))

contenido.append(Spacer(1, 0.5 \* cm))

# Tabla resumen

contenido.append(Paragraph(

"Resumen Financiero Mensual",

estilos["Heading2"]

))

contenido.append(Spacer(1, 0.3 \* cm))

tabla\_datos = [

["Concepto", "Monto (S/.)", "% del Ingreso"],

["Ingreso total", f"{datos['ingreso']:,.2f}", "100%"],

["Gastos necesarios", f"{datos['necesarios']:,.2f}", f"{datos['porc\_nec']:.1f}%"],

["Gastos discrecionales", f"{datos['discrecionales']:,.2f}", f"{datos['porc\_dis']:.1f}%"],

["Ahorro", f"{datos['ahorro']:,.2f}", f"{datos['porc\_aho']:.1f}%"],

["Balance", f"{datos['balance']:,.2f}", "---"],

]

tabla = Table(tabla\_datos, colWidths=[7\*cm, 4\*cm, 3\*cm])

contenido.append(tabla)

contenido.append(Spacer(1, 0.6 \* cm))

# Gráficos

contenido.append(Paragraph(

"Visualización de la Distribución Financiera",

estilos["Heading2"]

))

contenido.append(Spacer(1, 0.3 \* cm))

contenido.append(Image("barras.png", width=16\*cm, height=8\*cm))

contenido.append(Spacer(1, 0.4 \* cm))

contenido.append(Image("circular.png", width=16\*cm, height=8\*cm))

contenido.append(Spacer(1, 0.6 \* cm))

# Recomendaciones

contenido.append(Paragraph(

"Evaluación según la Regla 50-30-20",

estilos["Heading2"]

))

contenido.append(Spacer(1, 0.3 \* cm))

if datos["m\_nec"] >= 0:

contenido.append(Paragraph(

f"<b>Gastos Necesarios:</b> Dentro del rango. Margen disponible: S/. {datos['m\_nec']:,.2f}",

estilos["Normal"]

))

else:

contenido.append(Paragraph(

f"<b>Gastos Necesarios:</b> Exceden el límite en S/. {abs(datos['m\_nec']):,.2f}. Revise gastos fijos.",

estilos["Normal"]

))

if datos["m\_dis"] >= 0:

contenido.append(Paragraph(

f"<b>Gastos Discrecionales:</b> Controlados. Margen: S/. {datos['m\_dis']:,.2f}",

estilos["Normal"]

))

else:

contenido.append(Paragraph(

f"<b>Gastos Discrecionales:</b> Superan el límite en S/. {abs(datos['m\_dis']):,.2f}. Modere gastos.",

estilos["Normal"]

))

if datos["m\_aho"] <= 0:

contenido.append(Paragraph(

"<b>Ahorro:</b> Nivel adecuado según la regla 50-30-20.",

estilos["Normal"]

))

else:

contenido.append(Paragraph(

f"<b>Ahorro:</b> Incrementar al menos S/ {datos['m\_aho']:,.2f} para cumplir la meta.",

estilos["Normal"]

))

doc.build(contenido)

# Preparar datos y generar PDF

print("\n Generando reporte PDF...")

datos\_pdf = {

"ingreso": ingreso\_total,

"necesarios": gastos\_necesarios,

"discrecionales": gastos\_discrecionales,

"ahorro": ahorro,

"balance": balance,

"porc\_nec": porc\_necesarios,

"porc\_dis": porc\_discrecionales,

"porc\_aho": porc\_ahorro,

"m\_nec": margen\_necesarios,

"m\_dis": margen\_discrecionales,

"m\_aho": margen\_ahorro

}

generar\_reporte\_pdf(datos\_pdf)

print(" Reporte PDF generado: 'Reporte\_Financiero\_Personal.pdf'")