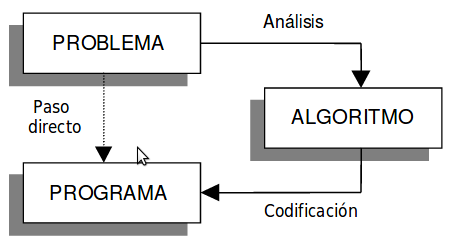
|  |  |
| --- | --- |
|  |  |
| ALGORITMOS |  |
|  | CURSO: SOFTWARE  ALUMNO: ARIEL CALDERÓN CUEVA DOCENTE: MSc. ENRIQUE BAÑO LEÓN |
|  |  |
|  |  |

# ¿Qué es un algoritmo?

Un algoritmo es un procedimiento por etapas. Es **un conjunto de reglas que hay que seguir para realizar una tarea** o resolver un problema.



Es **la capacidad de definir pasos claros** para resolver un problema.

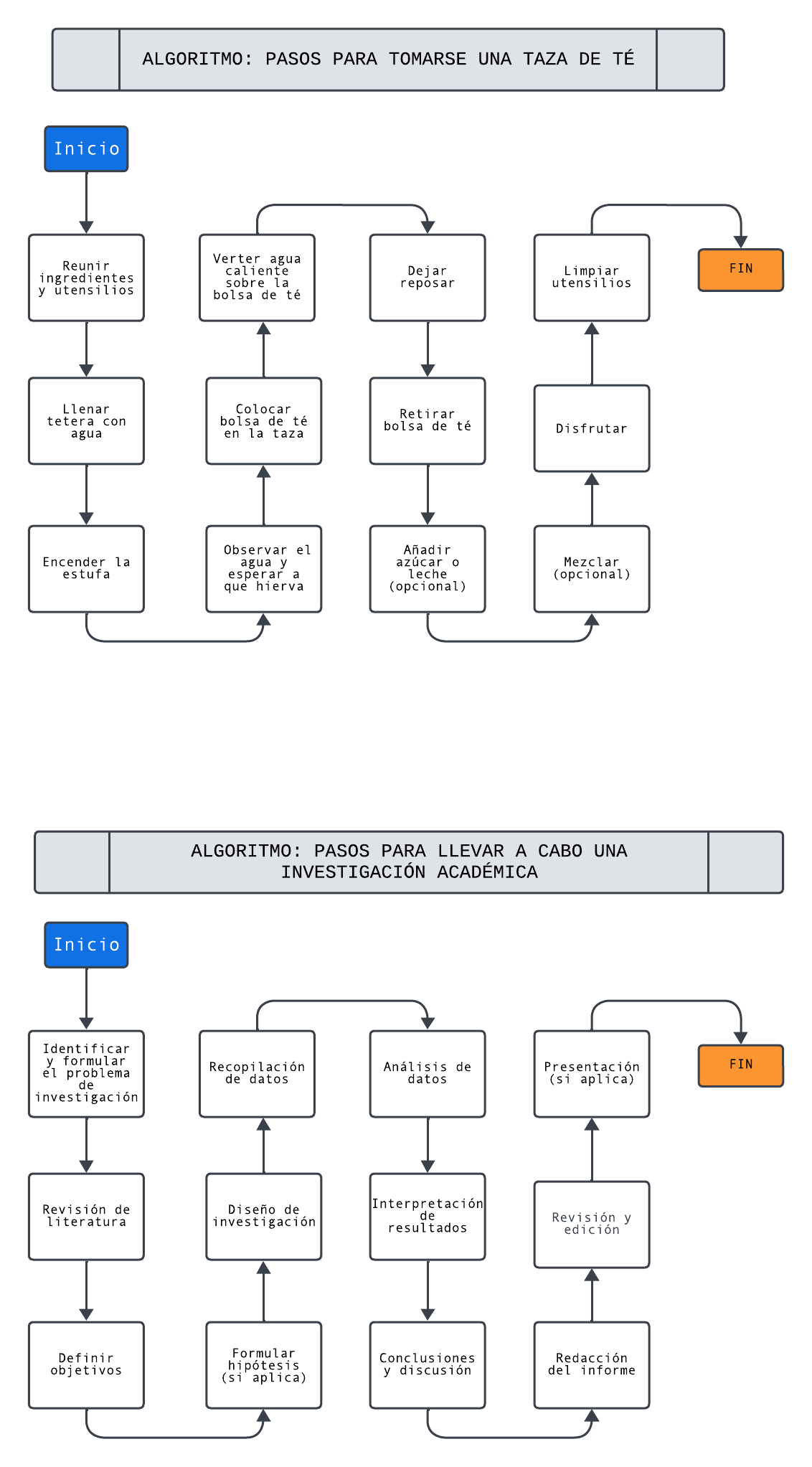
De hecho, utilizamos esta forma de pensar **a diario y a menudo sin darnos cuenta**. En la era de la **Data Science**, el **Machine Learning** y la **Inteligencia Artificial**, los algoritmos son más importantes que nunca y representan el combustible de la nueva revolución industrial.

En **el** **campo de la programación informática**, los algoritmos son conjuntos de reglas que indican al ordenador cómo ejecutar una tarea. En realidad, un programa informático es un algoritmo que indica al ordenador qué pasos debe realizar y en qué orden para llevar a cabo una tarea específica. Se escriben utilizando un lenguaje de programación.

### ¿Cuáles son los diferentes tipos de algoritmos?

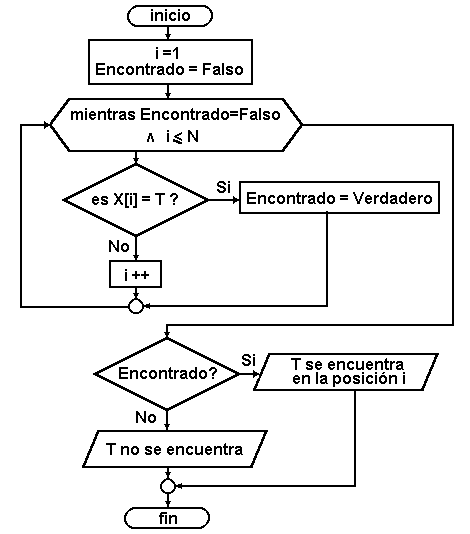
* Algoritmos de Búsqueda
* Algoritmos de Ordenamiento
* Algoritmos Recursivos
* Algoritmos de Grafos
* Algoritmos de Árboles
* Algoritmos de Caminos Mínimos
* Algoritmos de Flujo en Redes
* Algoritmos de Cadena de Texto
* Algoritmos de Machine Learning
* Algoritmos Genéticos

# Algoritmos cualitativos

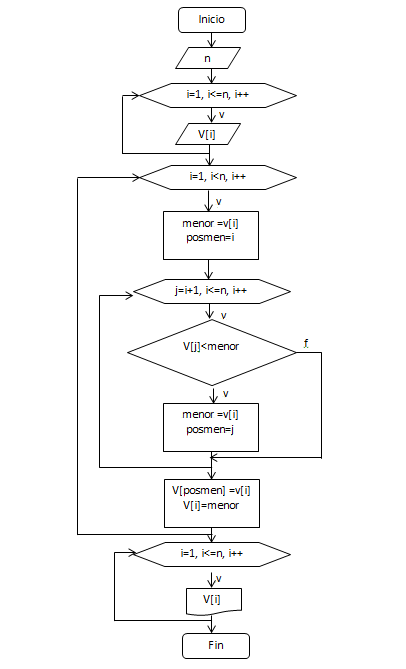


# Algoritmos cuantitativos

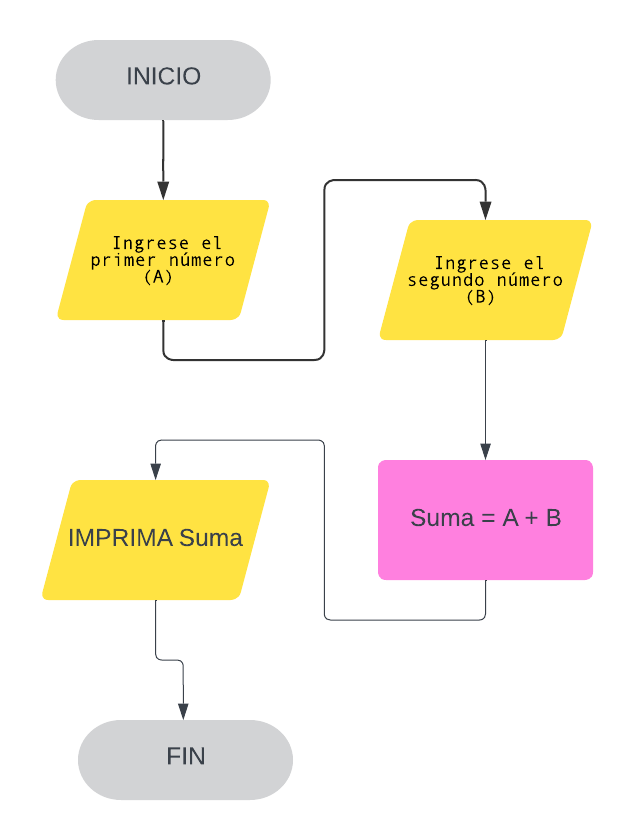
**ALGORITMO: BUSQUEDA SECUENCIAL**

****

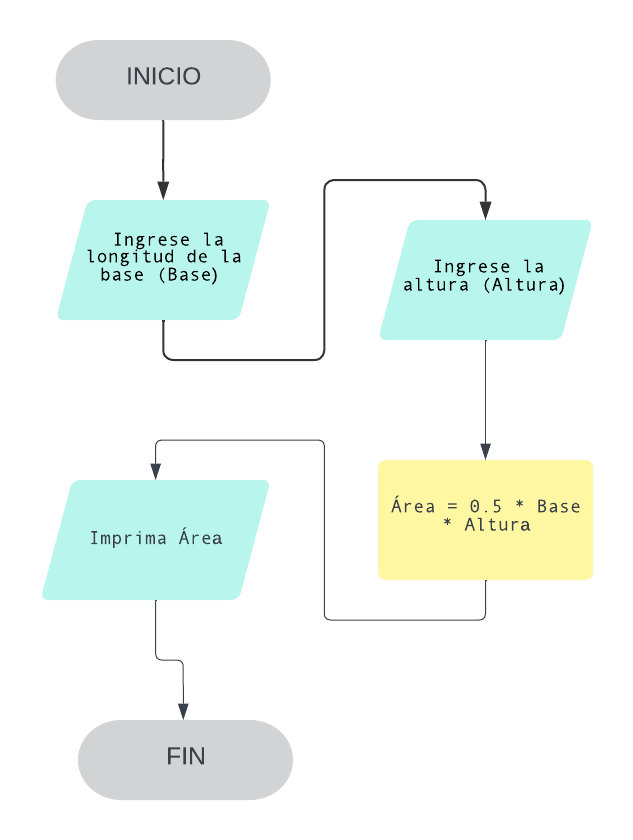
**ALGORITMO: ORDENAMIENTO DE UNA MATRIZ DE VALORES**

****

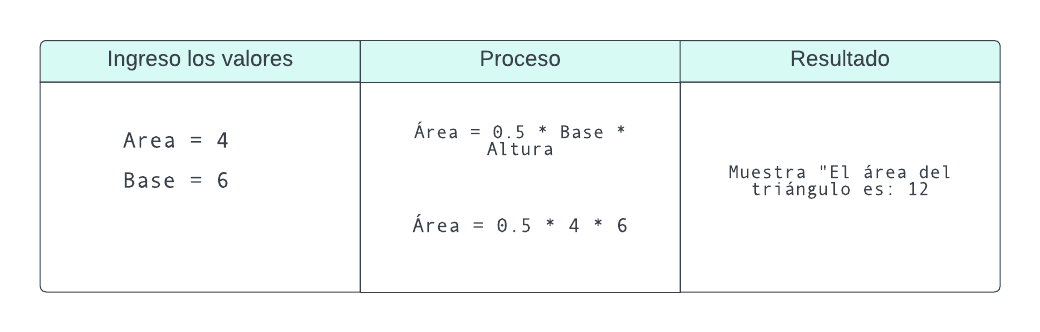
**Algoritmo: Suma de dos números**



**Algoritmo: Área de un triángulo**

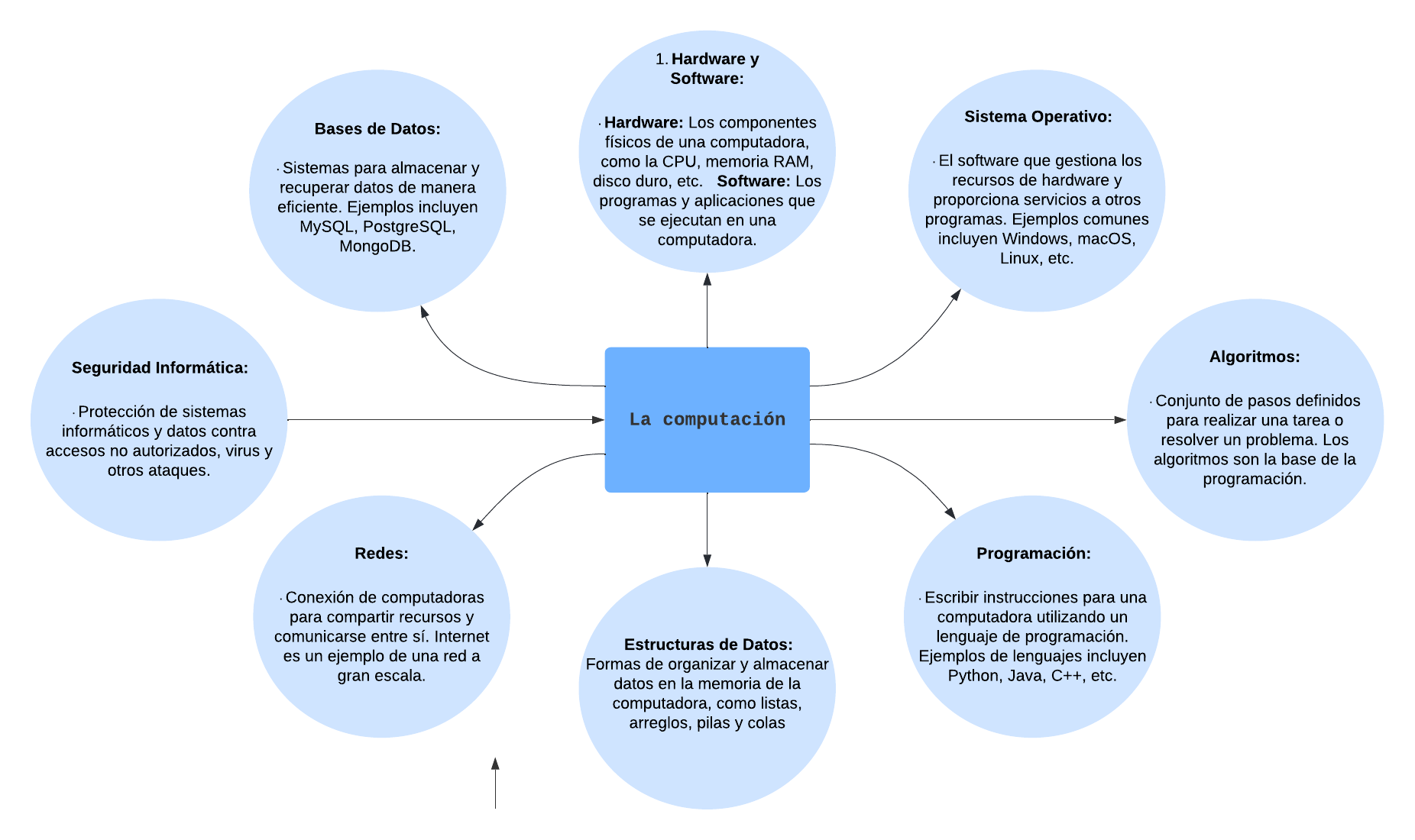


**Prueba de escritorio**



**LA COMPUTACIÓN, PROGRAMACIÓN Y PROGRAMAS**

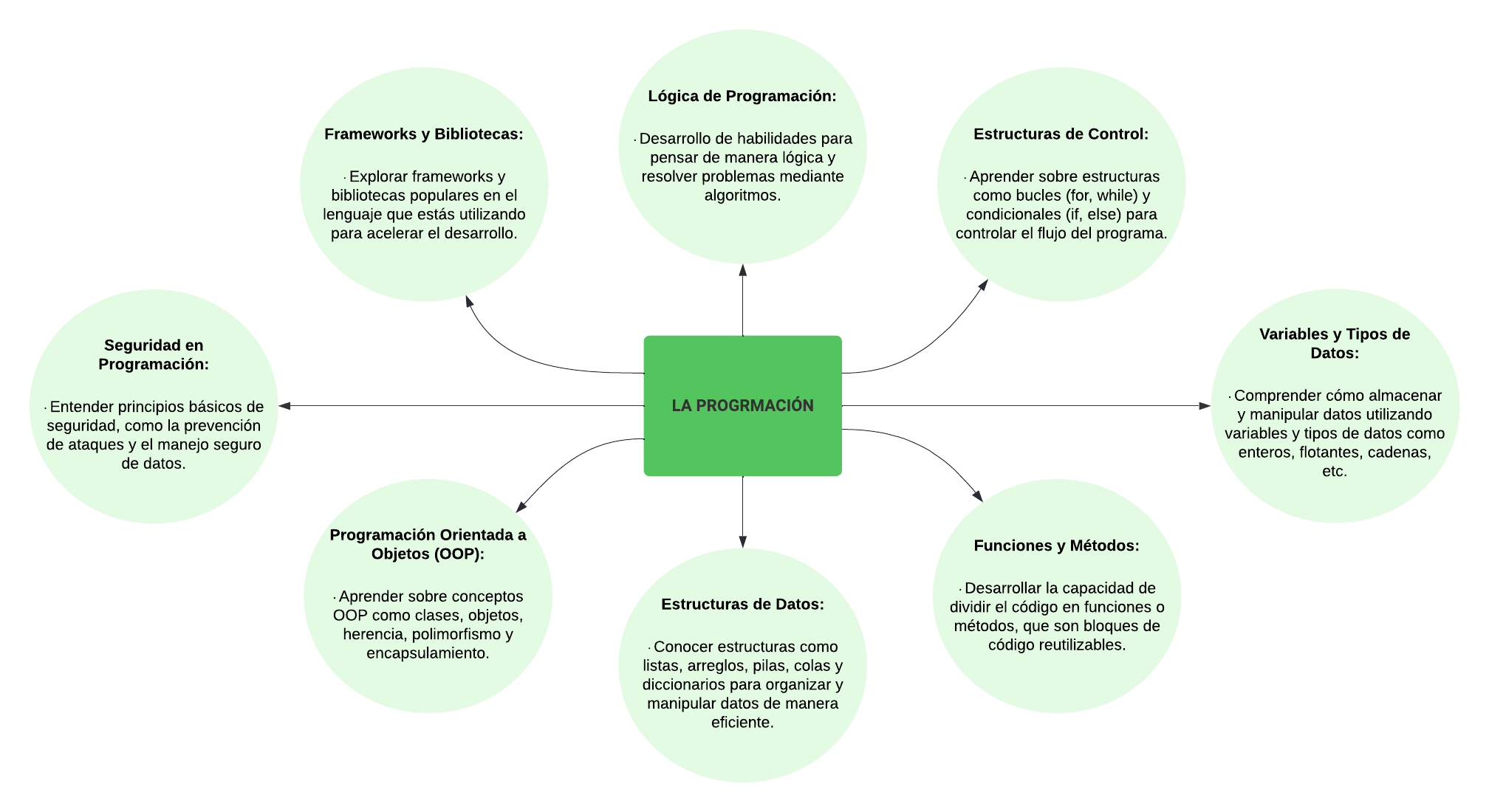
La computación es una disciplina que se ocupa del estudio y desarrollo de sistemas y programas informáticos, así como del procesamiento de información utilizando tecnologías de la computadora. Implica el diseño, la construcción y la utilización de sistemas informáticos y programas para resolver problemas, realizar tareas específicas y procesar datos.



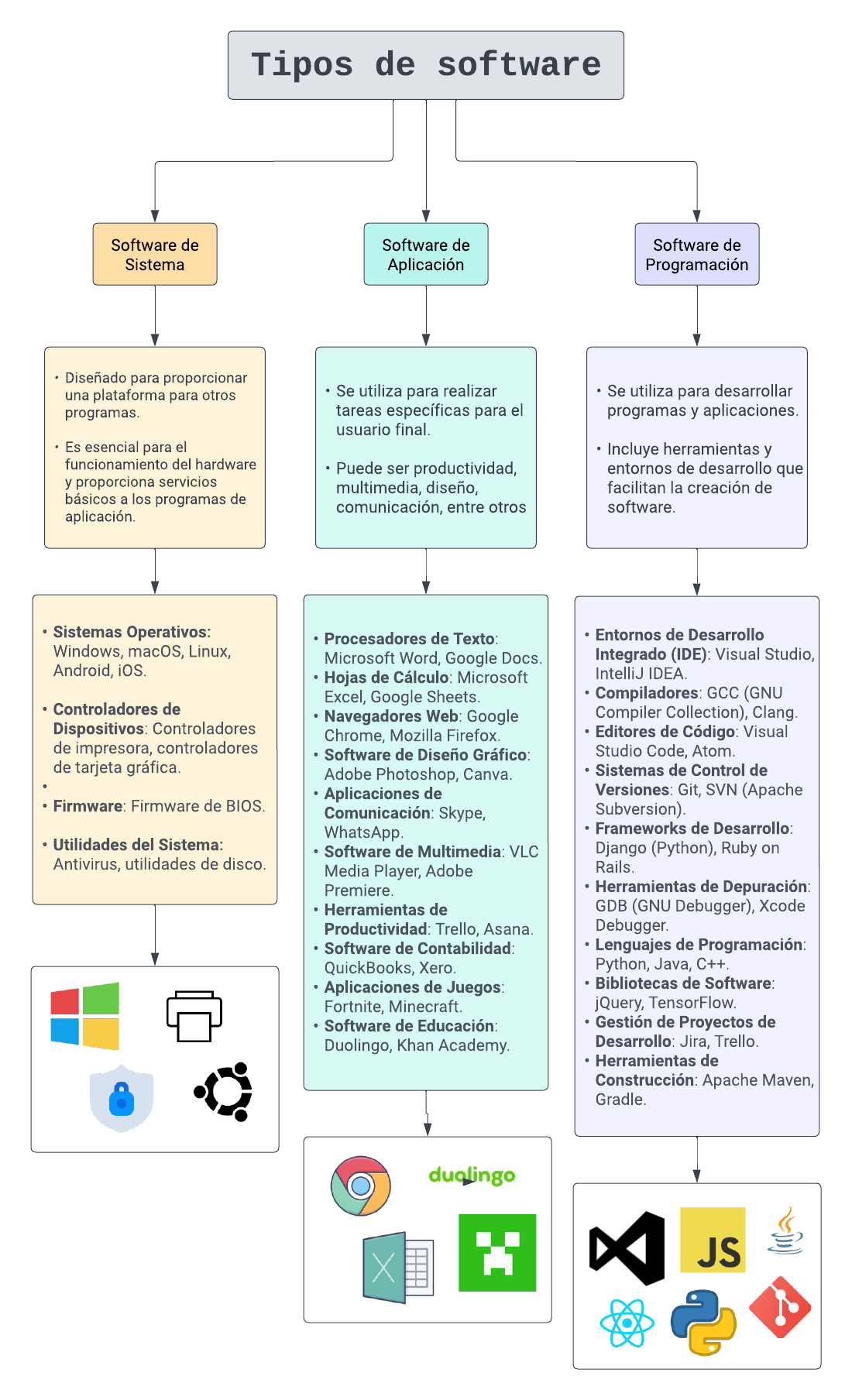
¿Qué es la programación?

La programación, en el contexto de la informática y la tecnología, se refiere al proceso de diseñar, codificar, depurar y mantener el código fuente de un programa de computadora. Un programa de computadora es un conjunto de instrucciones que le indican a una computadora cómo realizar una tarea específica. La programación es la habilidad de escribir estas instrucciones de manera clara y precisa para que la computadora las pueda ejecutar.





+



**Tipos de datos**

En el ámbito de la ciencia y la investigación, existen varios tipos de datos, y su importancia depende del contexto y del tipo de estudio.

**Tipos de Datos Numéricos:**

* **INT**: Números enteros.
* **DECIMAL** o **NUMERIC**: Números decimales o de punto flotante.
* **FLOAT**: Números de punto flotante de precisión simple.
* **DOUBLE**: Números de punto flotante de precisión doble.

**Tipos de Datos de Texto:**

* **CHAR(n)**: Cadenas de caracteres de longitud fija.
* **VARCHAR(n)**: Cadenas de caracteres de longitud variable.
* **TEXT**: Cadenas de texto de longitud variable (puede almacenar una cantidad mayor de caracteres que VARCHAR).

**Tipos de Datos de Fecha y Hora:**

* **DATE**: Fecha (año, mes, día).
* **TIME**: Hora del día.
* **DATETIME** o **TIMESTAMP**: Combinación de fecha y hora.

**Tipos de Datos Lógicos:**

* **BOOLEAN** o **BOOL**: Representa valores de verdad (verdadero o falso).
* **Tipos de Datos Binarios:**
* **BINARY(n)**: Datos binarios de longitud fija.
* **VARBINARY(n)**: Datos binarios de longitud variable.
* **BLOB**: Grandes objetos binarios, como imágenes o archivos.

**Tipos de Datos de Enumeración y Conjuntos:**

* **ENUM**: Lista de valores permitidos.
* **SET**: Conjunto de valores, donde se pueden seleccionar múltiples valores.
* **Tipos de Datos Especiales:**
* **NULL**: Representa la ausencia de un valor.
* **JSON**: Almacena datos en formato JSON.

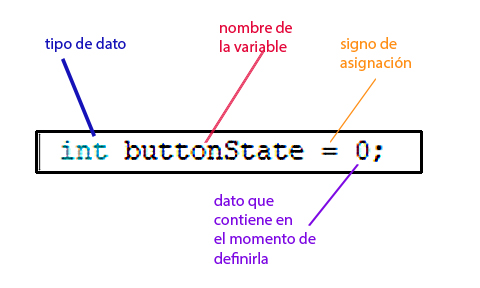
**Tipos de Datos Geoespaciales:**

Algunos sistemas de bases de datos, como PostgreSQL y MySQL con extensiones espaciales, admiten tipos de datos para almacenar y consultar información geoespacial.

|  |  |  |  |
| --- | --- | --- | --- |
| **Nombre** | **Memoria requerida** | **Rango** | **Descripción** |
| Booleano | 1bit | 1 ~ 0 | Verdad - Falso |
| Byte | 1 byte (8 Bits) | 0 ~ 255 | Byte sin signo. |
| ByteSig | 1 byte (8 Bits) | (-128) ~ 127 | Byte con signo. |
| Word | 2 byte (16 Bits) | 0 ~ 65.535 | Word sin signo. |
| WordSig | 2 byte (16 Bits) | (-32768) ~ 32767 | Word con signo. |
| Entero | 4 byte (32 Bits) | 0 ~ 4.294.967.295 | Entero sin signo. |
| EnteroSig | 4 byte (32 Bits) | (-2.147.483.648) ~ 2.147.483.647 | Entero con signo. |
| Real | 8 byte (64 Bits) | (-1,79769313486232^308) ~ (-4,94065645841247^-324) | Número con coma flotante de doble precisión |
| Decimal | 8 byte (64 Bits) | (-922.337.203.685.477,5800) ~ 922.337.203.685.477,5800 | Número con coma fija de 4 decimales. |
| Cadena | 1 byte por carácter | 0 ~ 2000 millones de caracteres | Cadena de caracteres alfanumérica. |

**Constantes & variables**

Las variables se almacenan en la memoria RAM y el espacio de memoria que ocupan (en bytes) depende de su tipo. Una constante tiene las mismas características que una variable excepto el hecho de que su valor asignado no puede ser cambiado durante la ejecución de programa.



**VARIABLES**

Cada variable debe ser declarada antes de ser utilizada en el programa. Como las variables se almacenan en la memoria RAM, es necesario reservar el espacio para ellas (uno, dos o más bytes). Al escribir un programa, usted sabe qué tipo de datos quiere utilizar y qué tipo de datos espera como resultado de una operación, mientras que el compilador no lo sabe. No se olvide de que el programa maneja las variables con los nombres asignados. El compilador las reconoce como números en la memoria RAM sin conocer su tamaño y formato. Para mejorar la legibilidad de código, las variables se declaran con frecuencia al principio de las funciones:

<tipo> variable;

Es posible declarar más de una variable de una vez si tienen el mismo tipo.

<tipo> variable1, variable2, variable3;

Aparte del nombre y del tipo, a las variables se les asignan con frecuencia los valores iniciales justamente enseguida de su declaración. Esto no es un paso obligatorio, sino ‘una cuestión de buenas costumbres’. Se parece a lo siguiente:

unsigned int peso; // Declarar una variable llamada peso

peso = 20; // Asignar el valor 20 a la variable peso

Un método más rápido se le denomina declaración con inicialización (asignación de los valores iniciales):

unsigned int peso = 20; // peso está declarado y su valor es 20

Si hay varias variables con el mismo valor inicial asignado, el proceso se puede simplificar:

unsigned int peso1 = peso2 = peso3 = 20;

int valor\_inicial = un\_mínimo\_de\_petróleo = 0;

**CONSTANTES**

Similar a las variables, las constantes deben ser declaradas antes de ser utilizadas en el programa. En mikroC, no es obligatorio especificar el tipo de constante al declararla. Por otra parte, las constantes deben ser inicializadas a la vez que se declaran. El compilador reconoce las constantes por su prefijo const utilizado en la declaración. Dos siguientes declaraciones son equivalentes:

const int MINIMUM = -100; // Declarar constante MINIMUM

const MINIMUM = -100; // Declarar constante MINIMUM

Las constantes pueden ser de cualquier tipo, incluyendo cadenas:

const T\_MAX = 3.260E1; // constante de punto flotante T\_MAX

const I\_CLASS = 'A'; // constante carácter I\_CLASS

const Mensaje = "Presione el botón IZQUIERDA"; // constante de cadena Mensaje

Las constantes de enumeración son un tipo especial de constantes enteras que hace un programa más comprensible al asignar los números ordinales a las constantes. Por defecto, el valor 0 se asigna automáticamente a la primera constante entre llaves, el valor 1 a la segunda, el valor 2 a la tercera etc.

enum surtidores {AGUA,GASÓLEO,CLORO}; // AGUA = 0; GASÓLEO = 1; CLORO = 2

Es posible introducir directamente el valor de una constante dentro de la lista de enumeraciones. El incremento se detiene al asignar un valor a un elemento de matriz, después se reinicia a partir del valor asignado. Vea el siguiente ejemplo:

enum surtidores {AGUA,GASÓLEO=0,CLORO}; // AGUA = 0; GÁSOLEO = 0; CLORO = 1

Las constantes de enumeración se utilizan de la siguiente manera:

int Velocidad\_de\_ascensor

enum motor\_de\_ascensor {PARADA,INICIO,NORMAL,MÁXIMO};

Velocidad\_de\_ascensor = NORMAL; // Velocidad\_de\_ascensor = 2

## **Variables**

void main() {

// Declaramos una variable para guardar la altura

double altura = 2.5;

// Declaramos una variable para guardar el número de ruedas

int ruedas = 4;

// Calculo el área de un rectángulo de base 3 metros y altura 2.5 metros

println(altura \* 3.0);

// Calculo lo que cuesta cambiar las cuatro ruedas si una vale 150?

println(ruedas \* 150);

}

**CONSTANTES**

void main() {

// Declaramos una constante

final double GRAVEDAD = 9.8;

// Variable con la altura en metros a la que está situado el objeto

double altura = 10;

// Cálculo del tiempo que tarda en caer

double tiempo = sqrt(2.0\*altura/GRAVEDAD);

println(tiempo);

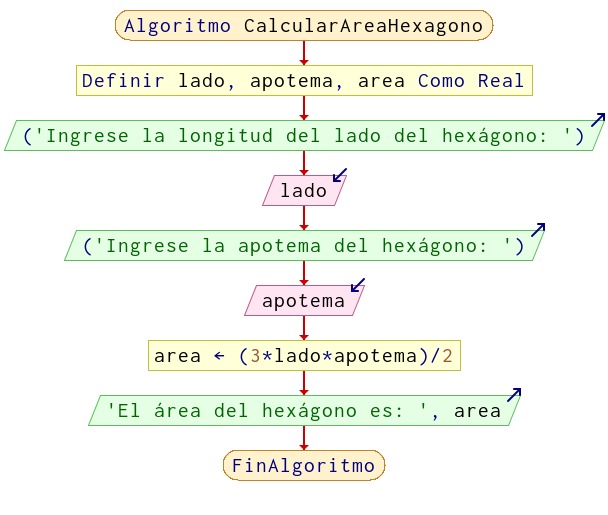
}

¿**Qué son los diagramas de flujo?**

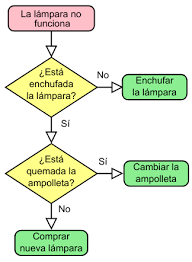
Un diagrama de flujo es un diagrama que describe un proceso, sistema o algoritmo informático. Se usan ampliamente en numerosos campos para documentar, estudiar, planificar, mejorar y comunicar procesos que suelen ser complejos en diagramas claros y fáciles de comprender

Los diagramas de flujo emplean rectángulos, óvalos, diamantes y otras numerosas figuras para definir el tipo de paso, junto con flechas conectoras que establecen el flujo y la secuencia. Pueden variar desde diagramas simples y dibujados a mano hasta diagramas exhaustivos creados por computadora que describen múltiples pasos y rutas

Los diagramas de flujo a veces se denominan con nombres más especializados, como diagrama de flujo de procesos, mapa de procesos o diagrama de flujo funcional.

****

****



**Las estructuras lógicas**  
  
Las estructuras lógicas son fundamentales en la programación y se utilizan para controlar el flujo de ejecución de un programa. Las principales estructuras lógicas son las condicionales y las estructuras de repetición.

Las utilizamos cuando en el desarrollo de la solución de un problema debemos tomar una decisión, para establecer un proceso o señalar un camino alternativo a seguir.

Estas estructuras lógicas son esenciales para controlar el flujo de ejecución en los programas y permiten que los desarrolladores implementen algoritmos más complejos y eficientes.

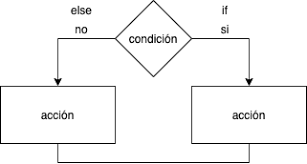


Estructuras Condicionales:

Las estructuras condicionales permiten que un programa tome decisiones basadas en una condición evaluada como verdadera o falsa. Comparan una variable contra otro(s) valor(es), para que con base al resultado de esta comparación, se siga un curso de acción dentro del programa. Cabe mencionar que la comparación se puede hacer contra otra variable o contra una constante, según se necesite.

En la mayoría de los lenguajes de programación, la estructura condicional más común es la declaración "if" (si), que se utiliza para ejecutar un bloque de código si una condición es verdadera.

También pueden incluir "else" (sino), que se ejecuta si la condición es falsa, y "else if" (sino si), que permite evaluar múltiples condiciones de manera secuencial.



Ejemplo en pseudocódigo:

si (condición) entonces

// código a ejecutar si la condición es verdadera

sino

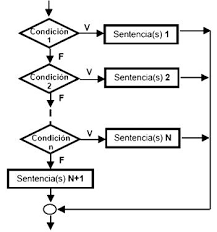
// código a ejecutar si la condición es falsa

fin si

Operadores lógicos (verdadero o falso):

Los operadores lógicos, como AND, OR y NOT, se utilizan para combinar y modificar condiciones en las estructuras condicionales.

Los operadores lógicos también se denominan operadores Booleanos. Devuelve True cuando Expresión1 y Expresión2 son verdaderas. Devuelve True cuando Expresión1 o Expresión2 es verdadera.



Ejemplo en pseudocódigo:

si (condición1 AND condición2) entonces

// código a ejecutar si ambas condiciones son verdaderas

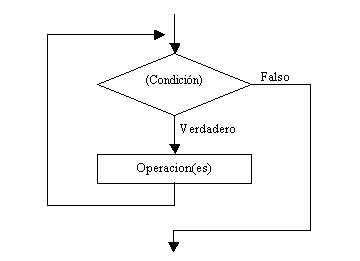
fin si

Estructuras de repetición

Se utilizan para realizar un determinado tipo de instrucciones en un número finito de veces. Se caracterizan por tener un punto inicial de partida, una condición la cual se encarga de ejecutar un número determinado de acciones hasta que esta condición no sea válida

Las estructuras de repetición más comunes son "for" y "while".

Un bucle "for" generalmente se utiliza cuando se conoce de antemano el número de iteraciones, mientras que un bucle "while" se utiliza cuando no se sabe cuántas veces se debe repetir el código.



Ejemplo en pseudocódigo de un bucle "for":

para i desde 1 hasta 10 hacer

// código a ejecutar en cada iteración

fin para

Ejemplo en pseudocódigo de un bucle "while":

mientras (condición) hacer

// código a ejecutar mientras la condición sea verdadera

fin mientras

**Identificadores**:

Los identificadores son nombres que se utilizan para identificar variables, funciones, clases u otros elementos en un programa. Deben seguir ciertas reglas, como comenzar con una letra, no contener espacios y ser significativos para facilitar la comprensión del código.

Los identificadores **son esenciales para referenciar** y manipular datos en un programa.

**Constantes:**

Las constantes **son valores que no cambian durante la ejecución de un programa**. Pueden ser numéricas, como 3.14, o cadenas de texto, como "Hola, mundo". El uso de constantes mejora la legibilidad del código y facilita la actualización de valores en un solo lugar.

**Operadores y Expresiones:**

Los operadores **son símbolos que realizan operaciones** sobre uno o más operandos. Las expresiones son combinaciones de valores y operadores que se evalúan para producir un resultado. Ejemplos de operadores incluyen aritméticos (+, -, \*, /), lógicos (&&, ||) y de comparación (==, !=).

**Abstracción de Datos:**

La abstracción de datos implica ocultar los detalles internos y **mostrar solo las características esenciales de un objeto**. En programación, las estructuras de datos y las clases permiten la abstracción de datos al encapsular información y proporcionar interfaces para interactuar con ella.

La abstracción de datos facilita el manejo de la complejidad al dividir un sistema en componentes más manejables.

**Variables Temporales:**

Las variables temporales son aquellas que se utilizan **para almacenar datos de manera temporal durante la ejecución de un programa**. Son útiles para realizar cálculos intermedios o almacenar valores temporales que se requieren en un momento específico. Las variables temporales ayudan a gestionar la manipulación de datos y facilitan la comprensión del flujo de un programa.

# Ejemplo de identificador para una variable

nombre\_usuario = "JohnDoe"

# Ejemplo de constantes

PI = 3.14159

SALUDO = "Hola, mundo"

# Ejemplo de operadores y expresiones

* a = 5 b = 2
* suma = a + b
* resta = a - b
* multiplicacion = a \* b
* division = a / b
* resultado\_logico = (a > b) and (a != 0)

# Ejemplo de abstracción de datos con una clase

class Persona:

def \_\_init\_\_(self, nombre, edad):

self.nombre = nombre

self.edad = edad

def obtener\_info(self):

return f"Nombre:{self.nombre},

Edad: {self.edad}"

# Uso de la clase

persona1 = Persona("Alice", 25)

print(persona1.obtener\_info())

construcción dE algoritmoS Y programas

**Estructuras Secuenciales**

Las estructuras secuenciales en programación se refieren a la ejecución de instrucciones en un orden específico, de manera secuencial. En algoritmos y programación, las estructuras secuenciales son la base fundamental para ejecutar tareas paso a paso. Las instrucciones se ejecutan una tras otra, en el orden en que aparecen. Se utilizan para realizar operaciones simples y lineales, como cálculos aritméticos, asignación de valores a variables y entrada/salida de datos. Este tipo de estructuras es esencial en el desarrollo de algoritmos y programas básicos.

**Construcción de Algoritmos**

La construcción de algoritmos se refiere al proceso de diseñar y crear secuencias de pasos lógicos para resolver un problema específico. Un algoritmo es un conjunto de instrucciones detalladas y organizadas que llevan a cabo una tarea o resuelven un problema. La construcción de algoritmos implica la identificación de pasos clave, la definición de variables, la toma de decisiones y la iteración. Algoritmos bien construidos son eficientes, claros y fácilmente comprensibles. Esta habilidad es esencial en la programación y la resolución de problemas computacionales.

**Diagramas de Flujo**

Los diagramas de flujo son representaciones gráficas de algoritmos o procesos, utilizando símbolos y líneas para ilustrar la secuencia de pasos. Estos diagramas proporcionan una forma visual de comprender la lógica y la estructura de un algoritmo antes de su implementación en un lenguaje de programación específico. Los símbolos comunes incluyen óvalos para representar el inicio/fin, rectángulos para procesos, rombos para decisiones, y flechas para indicar la dirección del flujo. Los diagramas de flujo son herramientas valiosas en el diseño y la comunicación de algoritmos entre programadores y otros profesionales.

**ANALISIS DE UN PROBLEMA**

**INTRODUCCIÓN**

La resolución de problemas mediante algoritmos es fundamental en el campo de la programación y la informática. En este contexto, el diseño de algoritmos eficientes y estructurados es esencial para abordar situaciones cotidianas que requieren una solución lógica y precisa. En este documento, se abordará el desarrollo de un algoritmo para calcular el cambio a entregar a un cliente, minimizando el número de monedas utilizadas. Siguiendo los pasos propuestos por el Msc. Marcelo Baño, se analizará el problema, se diseñará una solución lógica, se construirá un diagrama de flujo y se implementará el algoritmo en un formato cercano al código. Este proceso permitirá comprender la importancia de seguir una estructura definida para la resolución de problemas mediante algoritmos, brindando una base sólida para el desarrollo de soluciones informáticas efectivas.

**PROBLEMA** **PLANTEADO**

Disponemos de una máquina que puede dar cambio con 5 tipos de billetes distintos: 20, 10, 5, 2 y 1. Dado el precio del artículo y la cantidad entregada por el consumidor indicar el cambio a entregar empleando el menor número posible de monedas.

**ENTENDER EL PROBLEMA**

**• Define claramente el problema que estás tratando de resolver.**

En este caso, el problema es calcular la vuelta a entregar al cliente utilizando el menor número posible de monedas, dados el precio del artículo y la cantidad entregada por el consumidor.

**• Identifica las entradas (datos de entrada) y las salidas (resultados esperados).**

Valor de cantidad entregada y valor del objeto o articulo

Valores de salida serian la cantidad de billetes que hay que entregar por cada denominación.

**ANALISIS DEL PROBLEMA**

**• Descompón el problema en partes más pequeñas y manejables.**

En este caso, se puede dividir el problema en los siguientes pasos:

1. Calcular el cambio a entregar al cliente.
2. Determinar la cantidad de monedas necesarias para dar el cambio.
3. Seleccionar las monedas adecuadas para dar el cambio.

**• Identifica los pasos o procesos principales necesarios para resolver el problema.**

* Realizamos la resta para calcular el vuelto
* Seccionamos el problema para realizar una división entre cada billete y el vuelto
* El sobrante se seguirá dividiendo para los billetes hasta que nos quede un valor de cero

**DISEÑAR LA SOLUCIÓN**

**• Selecciona las estructuras de datos y algoritmos apropiados para resolver cada parte del problema.**

**Estructuras de datos**

* **Variables enteras:** Utiliza variables para almacenar el precio, la cantidad entregada, el vuelto y la cantidad de cada tipo de billete.

**Algoritmos:**

* **Resta simple:** Calcula el vuelto restando el precio de la cantidad entregada.
* **División entera y módulo:** Utiliza la división entera y el módulo (% en muchos lenguajes) para obtener la cantidad de billetes de cada denominación.
* **Bucle o estructuras condicionales:** Puedes utilizar un bucle o estructuras condicionales para verificar cuántos billetes de cada denominación son necesarios.

**• Decide sobre la secuencia de pasos que compondrán tu algoritmo.**

* Definir variables
* Entrada de datos
* Cálculo del vuelto
* Cálculo de billetes
* Salida de resultados

**ESCRIBIR EL PSEUDOCÓDIGO**

**•Traduce tu diseño a un formato más cercano al código:**

- Leer el precio del artículo y la cantidad entregada por el cliente. - Calcular el cambio a entregar al cliente. - Determinar la cantidad de monedas necesarias para dar el cambio. - Seleccionar las monedas adecuadas para dar el cambio. – Imprimir la cantidad de monedas necesarias para dar el cambio.

**REVISAR Y REFINAR**

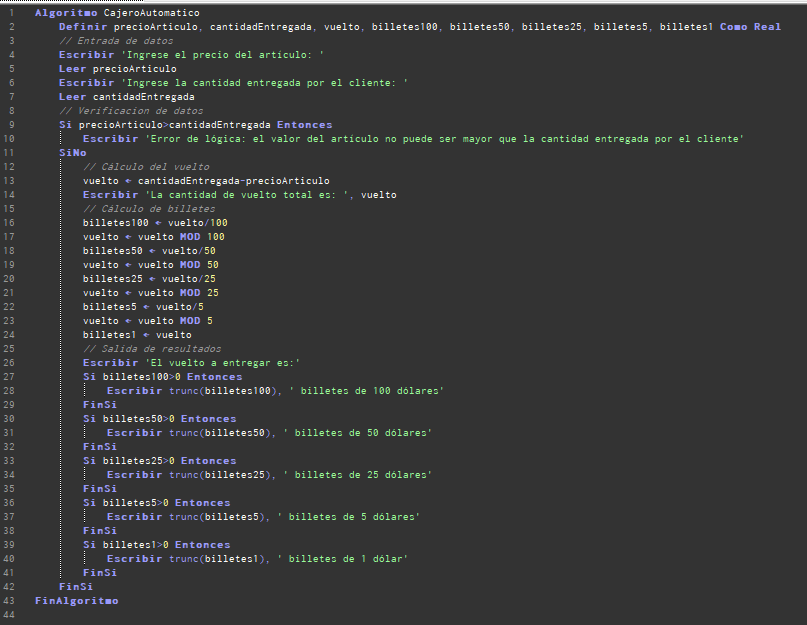
En nuestro caso, hubo que agregar ciertas funcionalidades:

* Una operación extra al momento de entregar el resultado ya que cada billete debería ser representado solo por enteros y no por sus decimales (derivados del vuelto que queda por calcular). La función **trunc( )** nos ayudara con el problema ya que elimina la parte decimal de un valor**.**
* Una verificación de datos ingresados. El valor del articulo **siempre será menor** a la cantidad entregada por el cliente, de lo contario no existiría vuelto alguno.

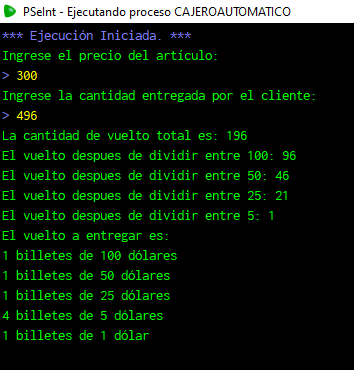
**CONSTRUIR EL DIAGRAMA DE FLUJO**



**IMPLEMENTAR EL CÓDIGO**



**PROBAR Y DEPURAR**



**Trabajo Grupal De Algoritmos y Lógica de Programación**

**Objetivo:** Crear un algoritmo el cual de solución al problema plantado

**Problema:** Una empresa paga a sus 100 operarios semanalmente, de acuerdo con el número de horas trabajadas, a razón de P pesetas por hora y de 1.5.P pesetas por cada hora extra. Las horas extras son las que exceden de 40h. Hacer el organigrama que dado el número de horas trabajadas y el valor de P calcule el salario que le corresponde.

1. **Entender el problema**

El problema planteado en general busca según la cantidad de horas trabajadas determinar el tipo de horas trabajadas por los trabajadores de una empresa y en base a ese dato determinar su salario en pesetas.

Las entradas de datos serian el nombre del trabajador, y el número de horas trabajadas, mientras que las salidas son el valor del salario; la variable es el número de horas trabajadas.

1. **Análisis del problema**

Las partes de problema son la lectura de datos de entrada, la estructura de datos, las fórmulas a usar, la salida de datos.

Dentro de la lectura de datos de entrada se pedirá que se ingrese su nombre y la cantidad de horas de trabajo, la estructura de datos a usar es la condicional, se va a usar dos formulas una si las horas de trabajo son menores o iguales a 40horas y otra si las horas de trabajo son mayores a 40, finalmente en la salida de datos se aplicara el comando escribir para mostrar el salario

1. **Diseñar la solución**

El primer paso para resolver este problema es la declaración de variables , después como segundo paso es necesario realizar el ingreso de las entradas de datos ; lo principal es determinar el número de horas trabajadas para lo cual utilizaremos la estructura de datos condicional ( si entonces) para determinar si el número de horas trabajadas se pueden considerar horas extras , si el número de horas trabajadas es menor a 40 horas se considerara al salario por la siguiente formula (salario= horas de trabajo \* 1) , mientras que si se exceden las 40 horas el salario se aplicara la siguiente formula( salario horas extras=horas de trabajo\*1.5) , según esta asignación se mostrara el salario correspondiente.

1. **Escribir el seudocódigo**

Algoritmo CalcularSalario

// Declaración de variables

Definir horas\_trabajadas, valor\_pesetas, salario\_base, horas\_extra, salario\_total Como Entero

// Solicitar al usuario el número de horas trabajadas

Escribir "Ingrese el número de horas trabajadas:"

Leer horas\_trabajadas

// Solicitar al usuario el valor de Pesetas

Escribir "Ingrese el valor de Pesetas por hora:"

Leer valor\_pesetas

// Verificar si las horas trabajadas son menores o iguales a 40

Si horas\_trabajadas <= 40 Entonces

// Calcular el salario base sin horas extra

salario\_base = horas\_trabajadas \* valor\_pesetas

// No hay horas extra trabajadas

horas\_extra = 0

Sino

// Calcular el salario base considerando 40 horas

salario\_base = 40 \* valor\_pesetas

// Calcular las horas extra trabajadas

horas\_extra = horas\_trabajadas - 40

FinSi

// Calcular el salario total considerando las horas extra

salario\_total = salario\_base + (horas\_extra \* 1.5 \* valor\_pesetas)

// Mostrar el salario total

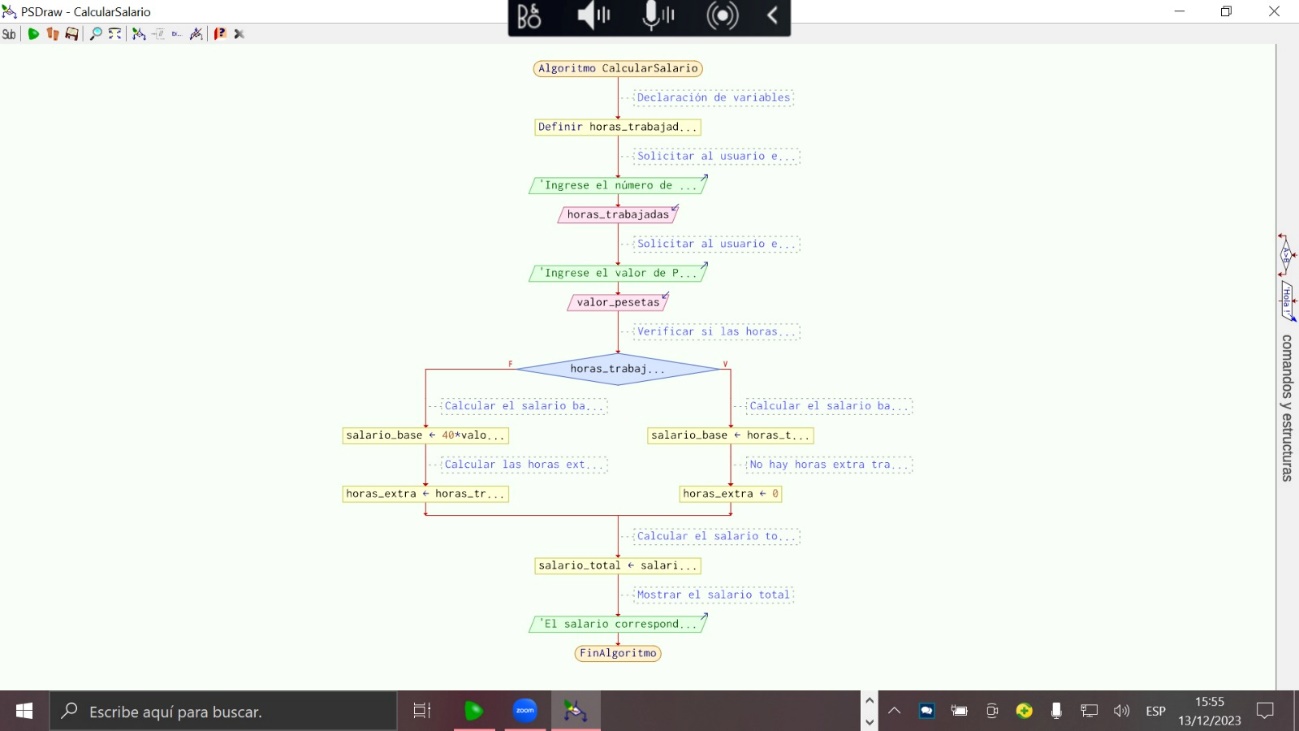
Escribir "El salario correspondiente es:", salario\_total

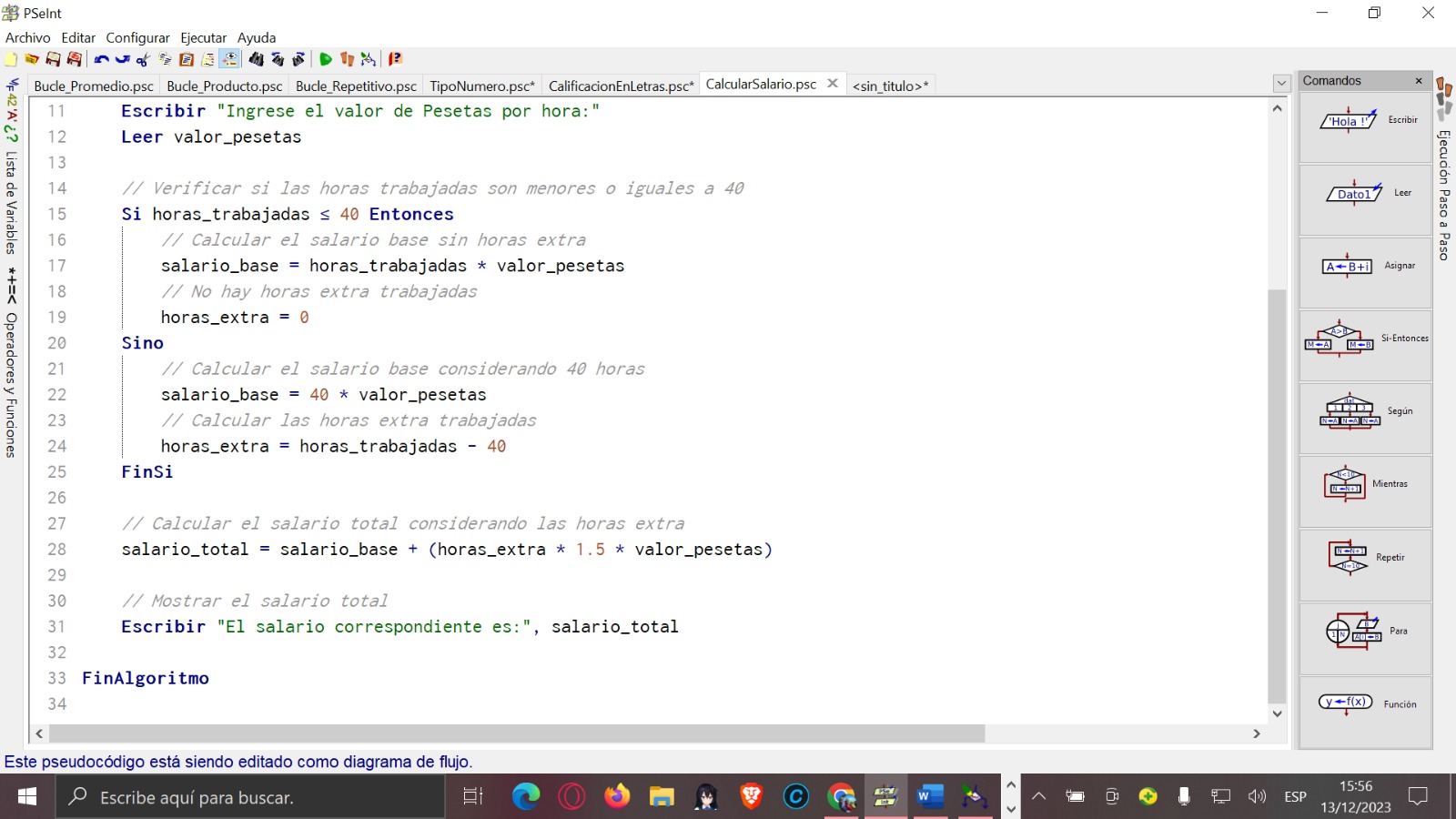
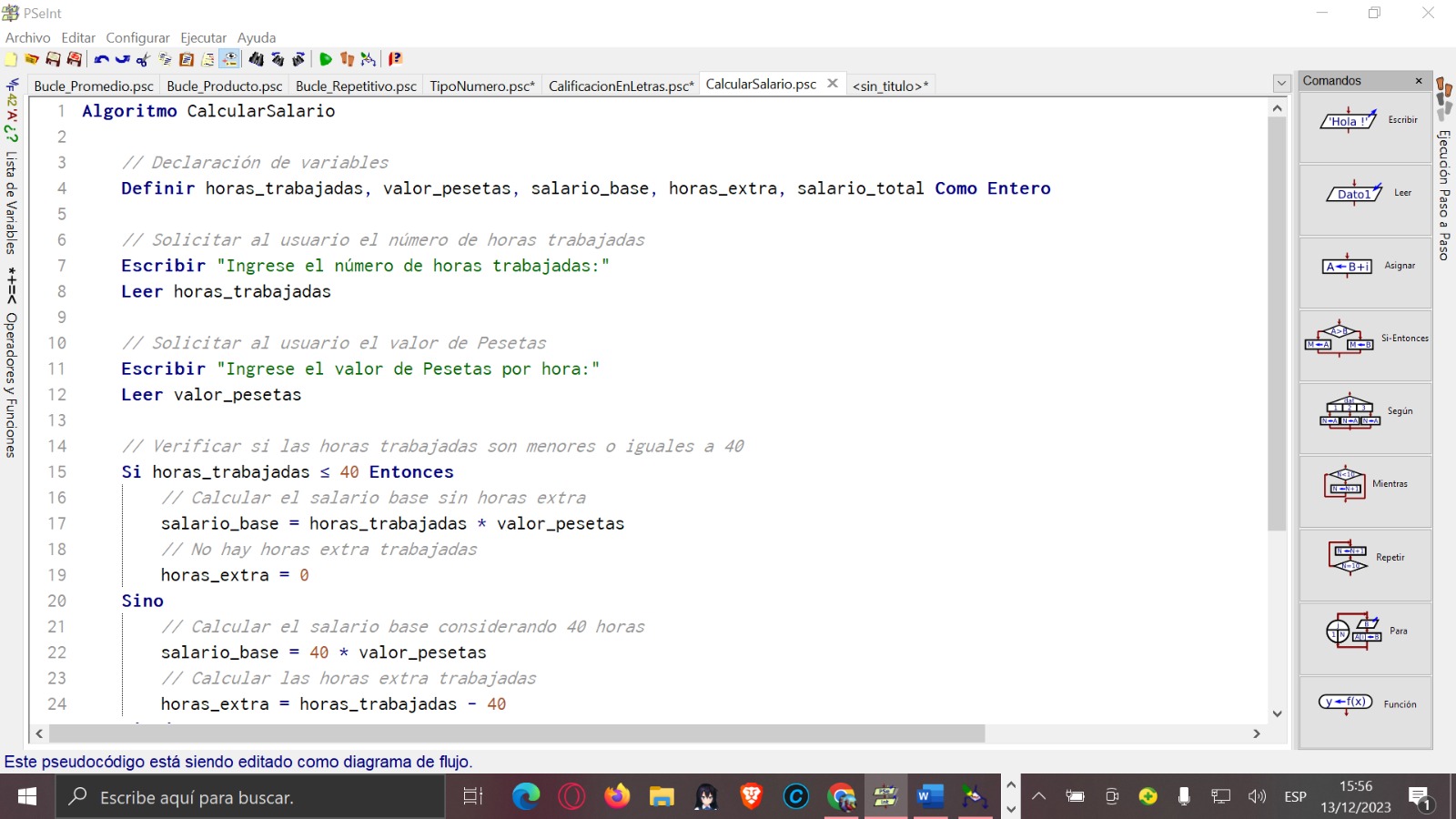
FinAlgoritmo

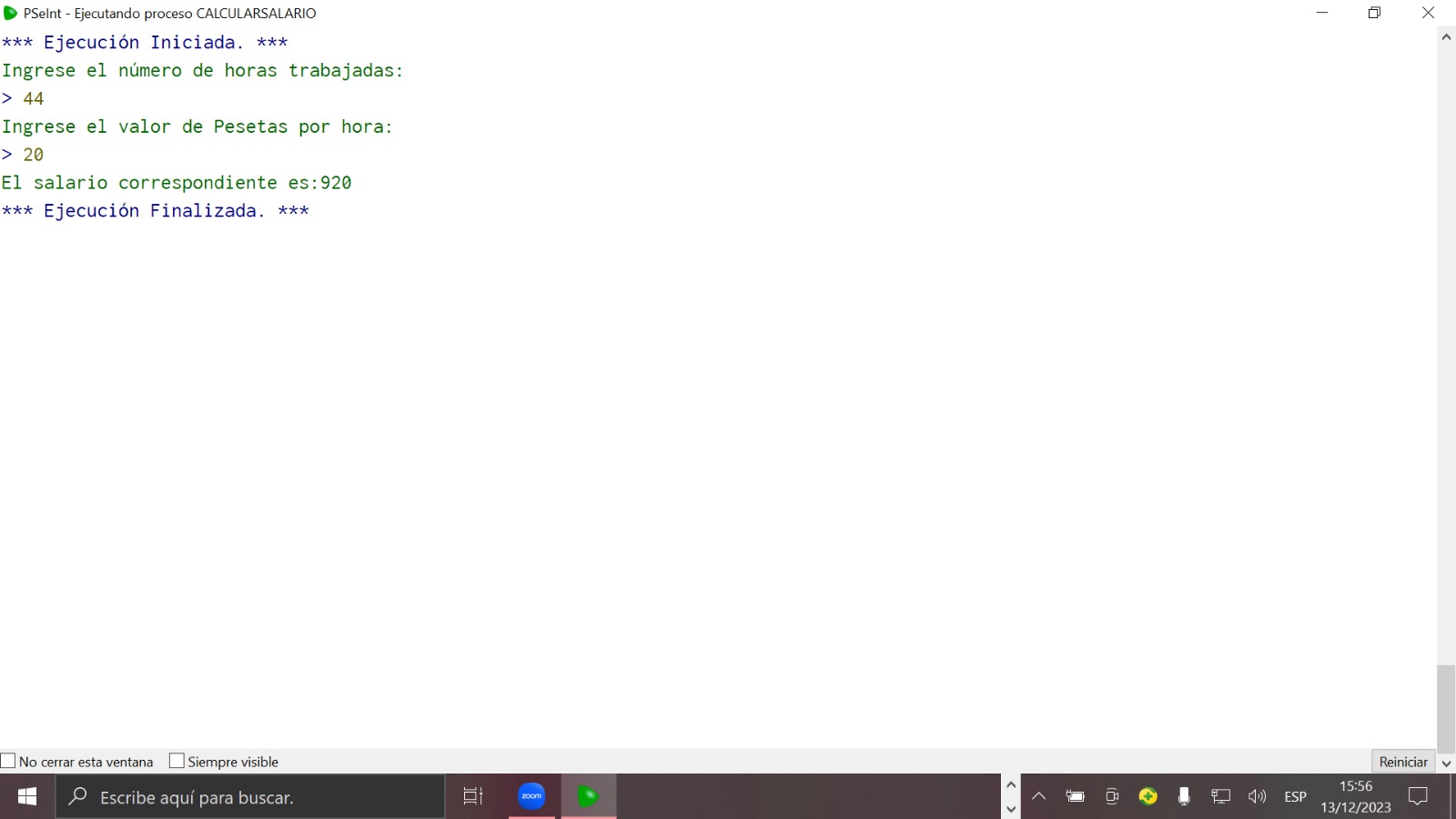
1. **Revisar y refinar**

En esta parte se reviso que todas las instrucciones de este algoritmo se encuentren correctamente ordenadas, además que tengan todos los signos correctamente ubicados, que no haya faltas ortográficas en la redacción de los procesos y que el algoritmo cumpla lo que solicita el problema planteado.

1. **Construir el diagrama de flujo**



1. **Implementar el código**



1. **Probar y depurar**

**METOLOGIA DE RESOLUCION DE PROBLEMAS**

**INTRODUCCIÓN**

La resolución efectiva de problemas es crucial en diversos contextos, tanto personales como profesionales. La metodología para la solución de problemas propuesta por el Msc. Marcelo Baño León ofrece un enfoque estructurado y eficaz para abordar estos desafíos. En este documento, se presenta un algoritmo desarrollado siguiendo dicha metodología, el cual resuelve el desafío de determinar la edad de las personas a partir del año de nacimiento, ilustrando la aplicabilidad y relevancia de esta metodología en situaciones cotidianas.

**PROBLEMA** **PLANTEADO**

Una empresa que contrata personal requiere determinar la edad de las personas que solicitan trabajo, pero cuando se les realiza la entre- vista sólo se les pregunta el año en que nacieron. Realice el pseudocódigo que representen el algoritmo para solucionar este problema.

**1. Comprender claramente cuál es el problema:**

La empresa necesita determinar la edad de las personas que solicitan trabajo, pero solo se les pregunta el año en que nacieron. Por lo tanto, se requiere un algoritmo que permita calcular la edad a partir del año de nacimiento proporcionado.

2. **Formular preguntas clave para entender mejor la situación:**

- ¿Qué información se necesita para calcular la edad de una persona?

Su año de nacimiento y el año actual

- ¿Cómo se puede obtener el año actual?

Alguna funcion prediseñada del programa que nos proporcione la fecha actual.

- ¿Cómo se puede calcular la edad a partir del año de nacimiento?

Restando el año actual menos el año de nacimiento del candidato.

3. **Identificar los factores relevantes y las restricciones del problema:**

- Se necesita el año de nacimiento de la persona.

- Se requiere el año actual para calcular la edad.

- La edad se puede calcular restando el año de nacimiento del año actual.

- El algoritmo debe ser capaz de manejar diferentes años de nacimiento y calcular la edad correspondiente.

4. **Desarrollo del algoritmo:**

A partir de la información obtenida en los pasos anteriores, se puede desarrollar un algoritmo que permita calcular la edad de una persona a partir del año de nacimiento proporcionado. El pseudocódigo del algoritmo es el siguiente:

Inicio

Definir variables

Entero añoNacimiento, añoActual, edad

Solicitar el año de nacimiento

Escribir "Ingrese el año de nacimiento:"

Leer añoNacimiento

Obtener el año actual

añoActual = ObtenerAñoActual()

Calcular la edad

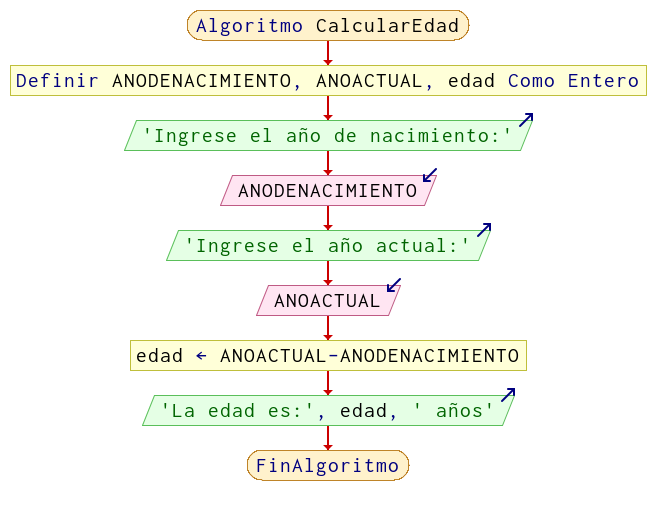
edad = añoActual - añoNacimiento

Mostrar la edad

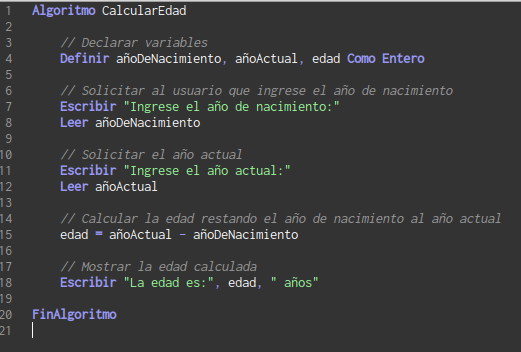
Escribir "La edad de la persona es:", edad

Fin

**5. Diagrama de flujo:**



**6. Implementación de código**

****

**Conclusiones**

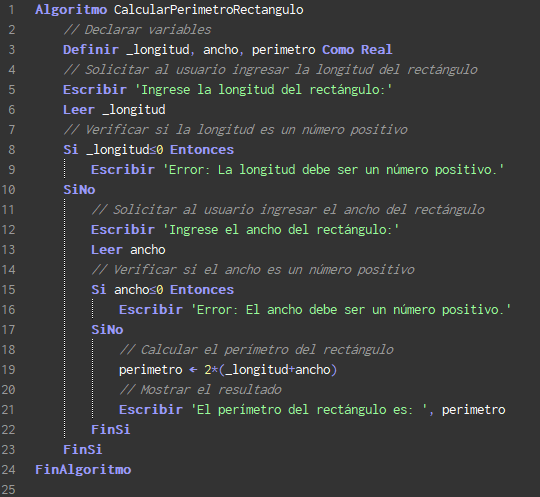
Se aplicó esta metodología para desarrollar un algoritmo que permite calcular la edad de una persona a partir del año de nacimiento proporcionado. El algoritmo desarrollado es sencillo y eficiente, y puede ser utilizado por la empresa para determinar la edad de las personas que solicitan trabajo.

**EJERCICIOS DE RESOLUCIÓN DE PROBLEMAS**

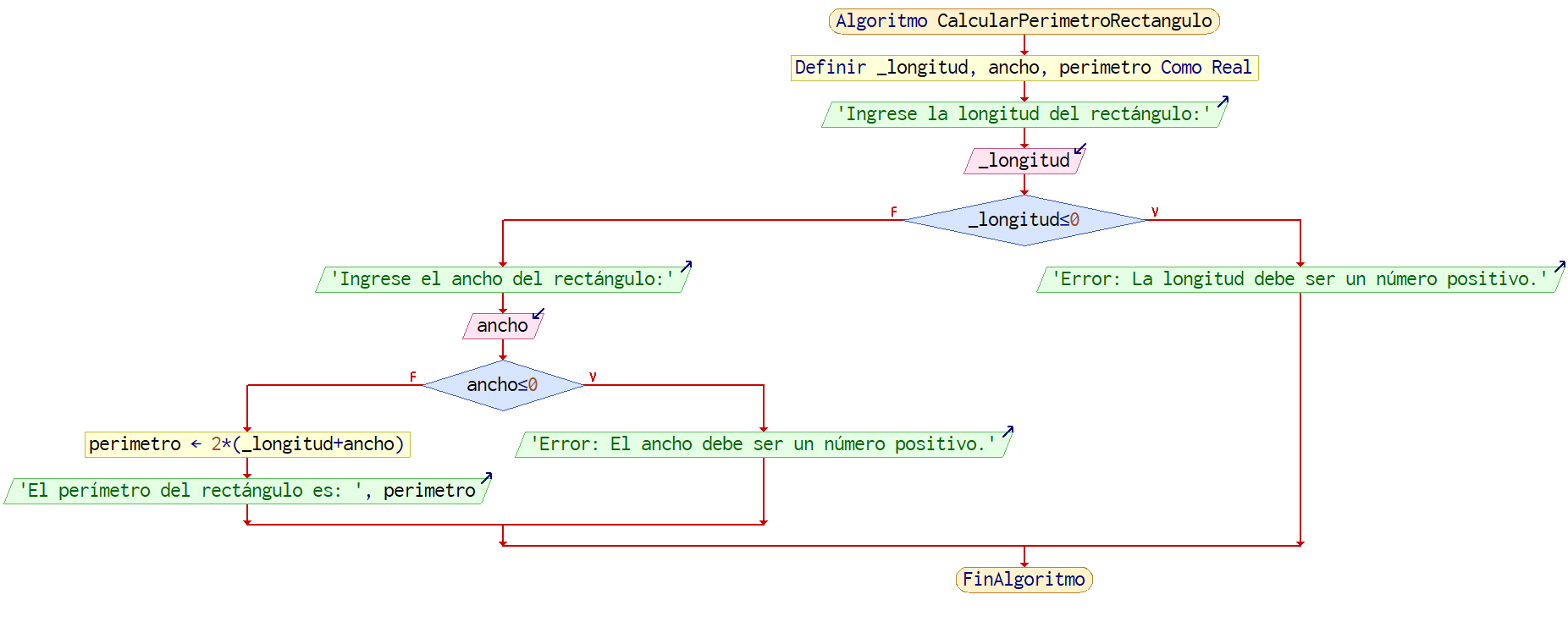
**Problema: Calcular el perímetro de un rectángulo**

Diseña un programa que solicite al usuario ingresar la longitud y el ancho de un rectángulo. Asegúrate de validar que ambos valores sean números positivos. Si el usuario ingresa un valor no válido, muestra un mensaje de error. Si los valores son válidos, calcula el perímetro del rectángulo utilizando la fórmula 2×(longitud+ancho) y muestra el resultado.

**Implementacion**:



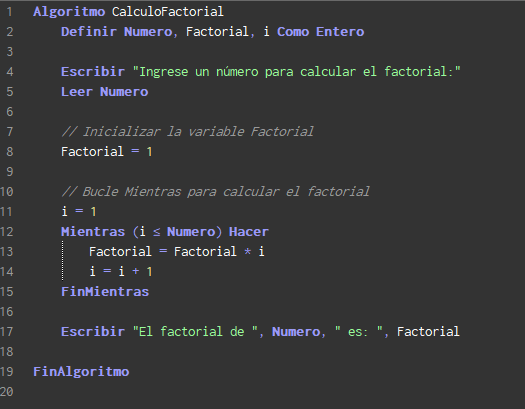
**Diagrama de flujo:**



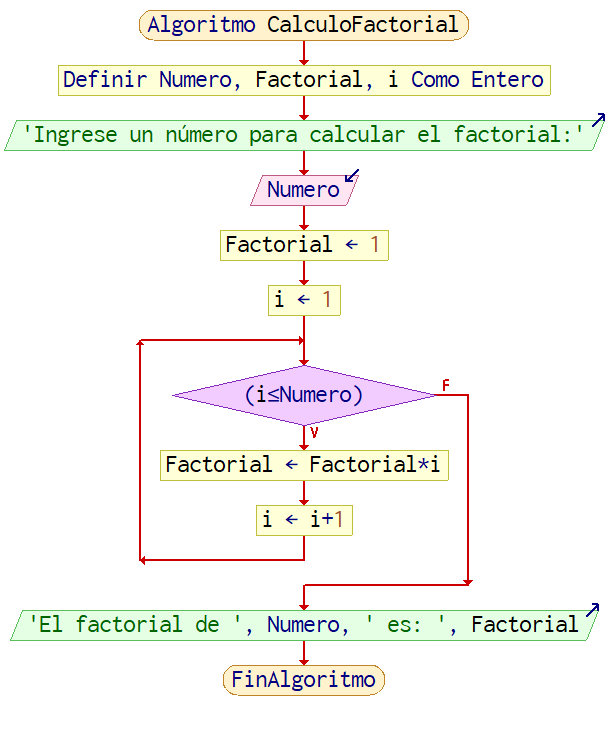
**Problema: Calcular del factorial**

Diseña un programa que solicite al usuario ingresar un número entero no negativo y calcule su factorial. Asegúrate de manejar correctamente los casos en que el usuario ingrese un número negativo. Si el usuario proporciona un número negativo, muestra un mensaje de error indicando que el factorial no está definido para números negativos. Si el usuario ingresa un número no negativo, calcula el factorial utilizando un bucle mientras y muestra el resultado.

**Implementacion**:



**Digrama de flujo:**

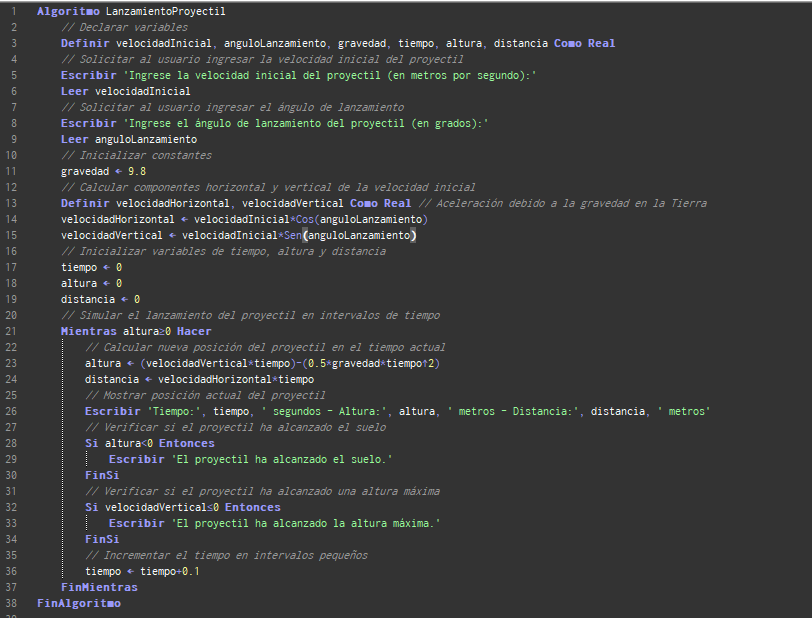


**Problema: Simulacion de lanzamiento de proyectil**

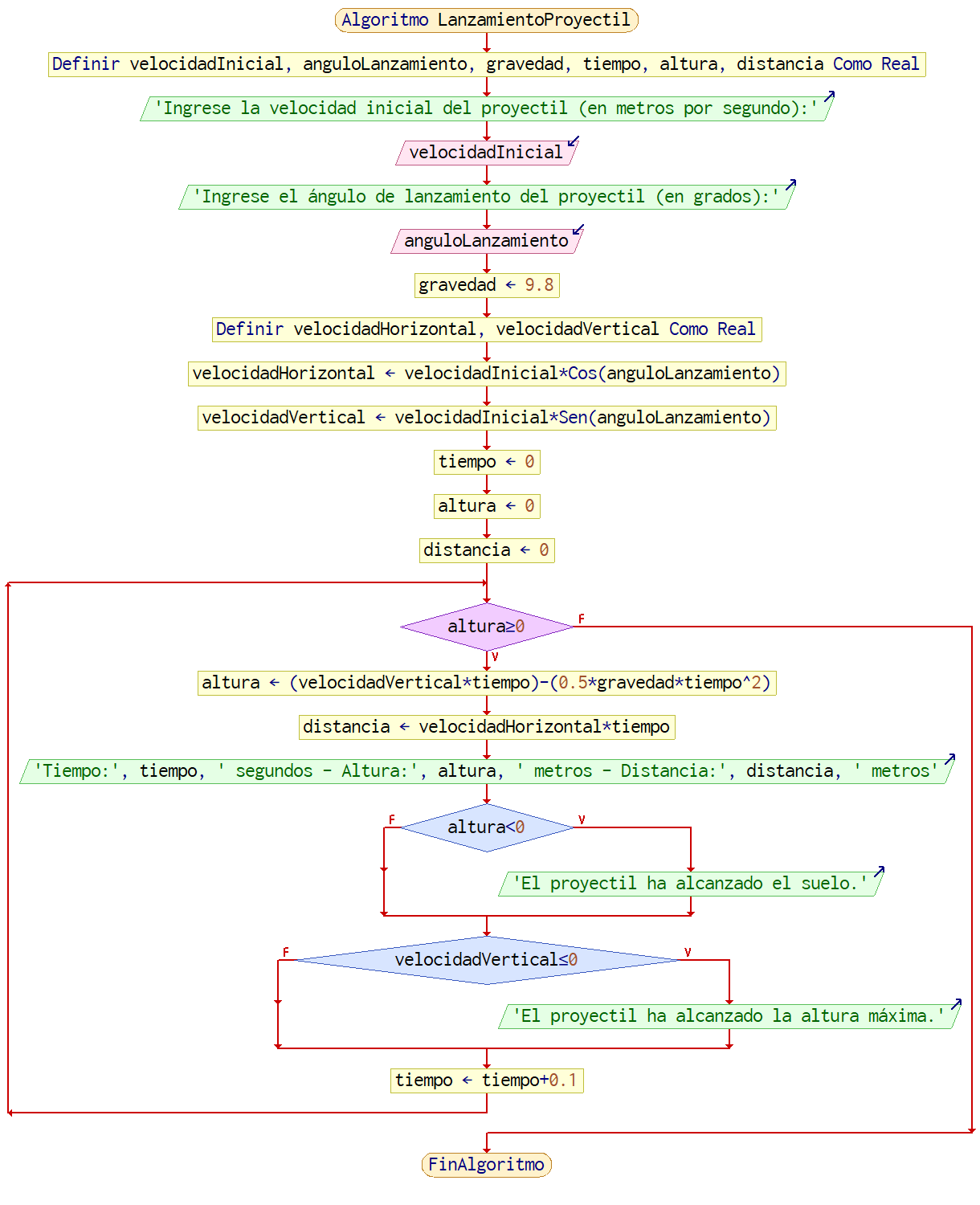
Diseña un programa que simule el lanzamiento de un proyectil. El usuario debe ingresar la velocidad inicial del proyectil en metros por segundo y el ángulo de lanzamiento en grados. El programa calculará y mostrará la posición del proyectil en intervalos de tiempo, incluyendo la altura y la distancia recorrida. La simulación deberá detenerse y mostrar un mensaje cuando el proyectil alcance el suelo o alcance su altura máxima.

Asegúrate de manejar correctamente las unidades y de informar al usuario sobre la finalización de la simulación, ya sea porque el proyectil alcanzó el suelo o su altura máxima.

**Implementacion**:



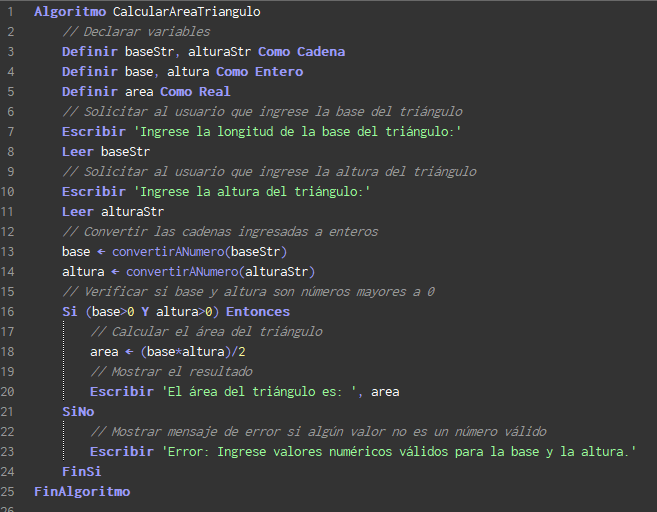
**Digrama de flujo:**



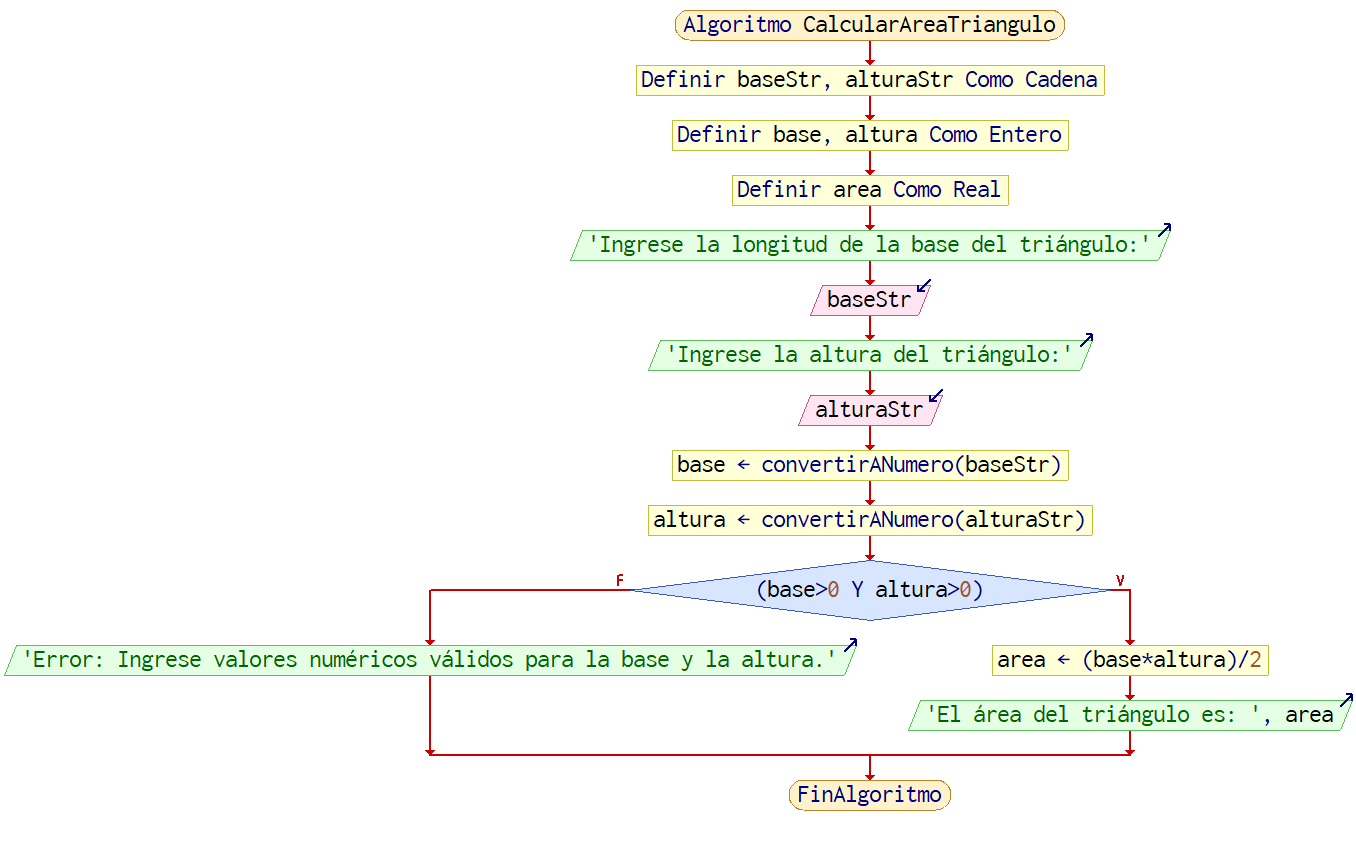
**Problema: Calcular el área de un triángulo**

Diseña un programa que solicite al usuario ingresar la longitud de la base y la altura de un triángulo. El programa debe validar que ambos valores sean números positivos. Si el usuario ingresa valores no válidos, muestra un mensaje de error indicando que se deben ingresar valores numéricos válidos para la base y la altura. Si los valores son válidos, calcula el área del triángulo utilizando la fórmula Area = (base × altura) / 2 y muestra el resultado.

**Implementacion**:



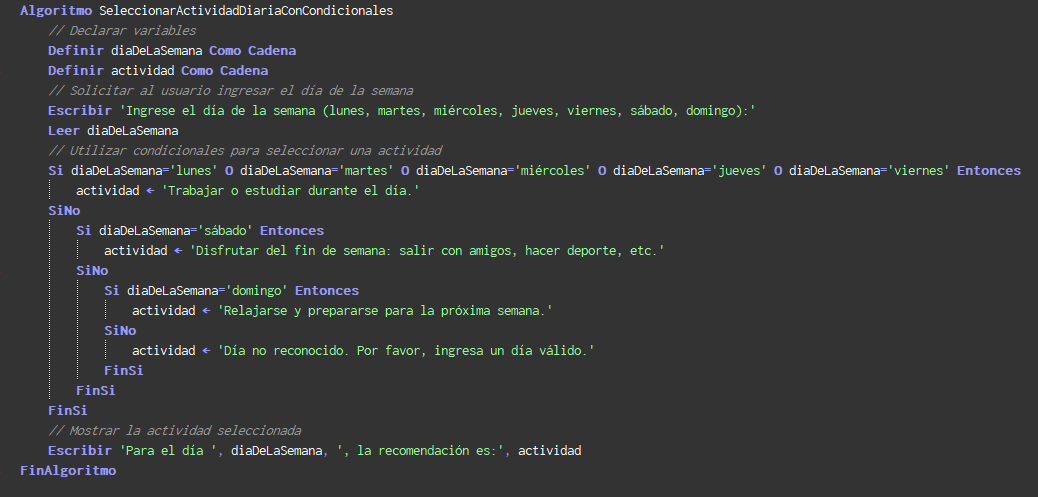
**Digrama de flujo:**



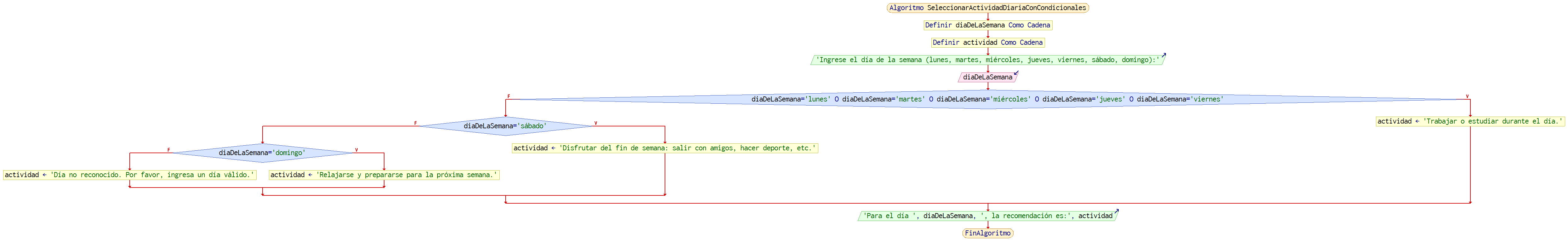
**Problema: Planificacion de actividades**

Diseña un programa que ayude a una persona a planificar sus actividades diarias para la semana. El usuario debe ingresar el día de la semana, y el programa le proporcionará una recomendación sobre qué actividad realizar con base en el algoritmo implementado.

**Implementacion**:



**Digrama de flujo:**

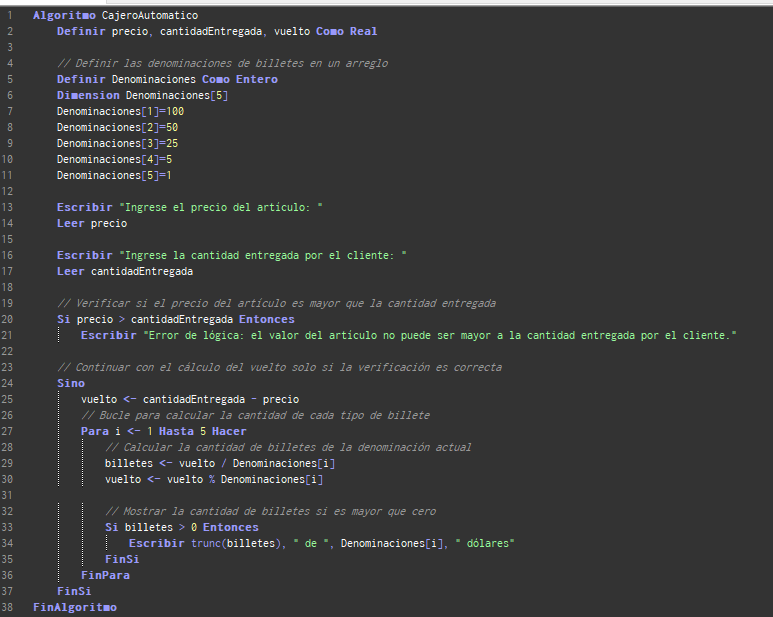


**Problema: Cajero de tienda**

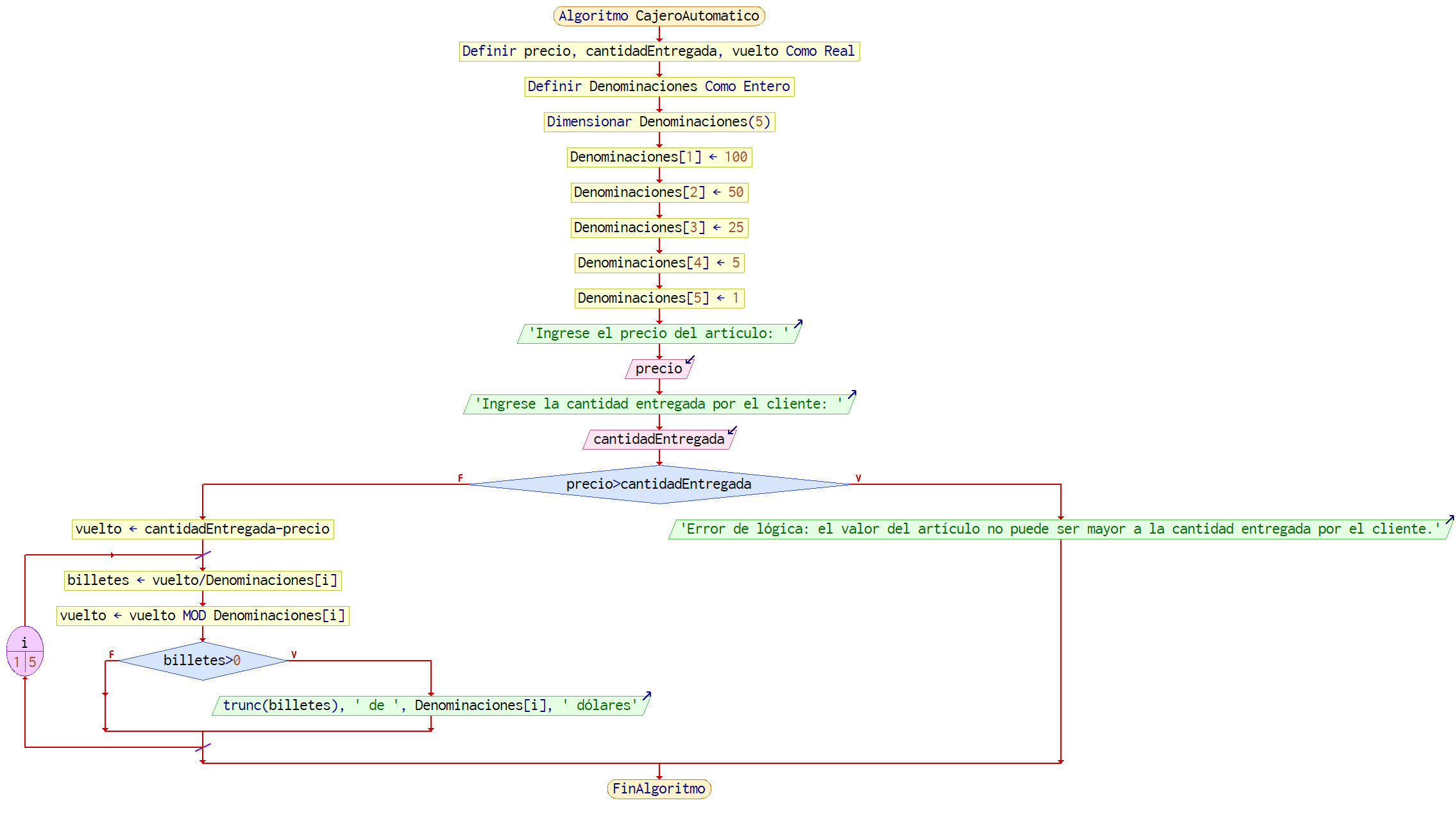
Una tienda ha implementado un cajero automático para agilizar el proceso de pago de sus clientes. El cajero debe calcular el vuelto a entregar al cliente utilizando las denominaciones de billetes disponibles (100, 50, 25, 5, 1 dólares). Escribe un algoritmo que permita al cajero automatizado realizar el cálculo del vuelto de la siguiente manera:

* Solicita al usuario ingresar el precio total de los artículos comprados.
* Pide al usuario ingresar la cantidad entregada por el cliente.
* Verifica que el precio del artículo no sea mayor que la cantidad entregada. En caso de que lo sea, muestra un mensaje de error y finaliza el programa.
* Si el precio es menor o igual a la cantidad entregada, calcula y muestra al cliente el vuelto utilizando el menor número posible de billetes en las denominaciones mencionadas.

**Implementacion**:



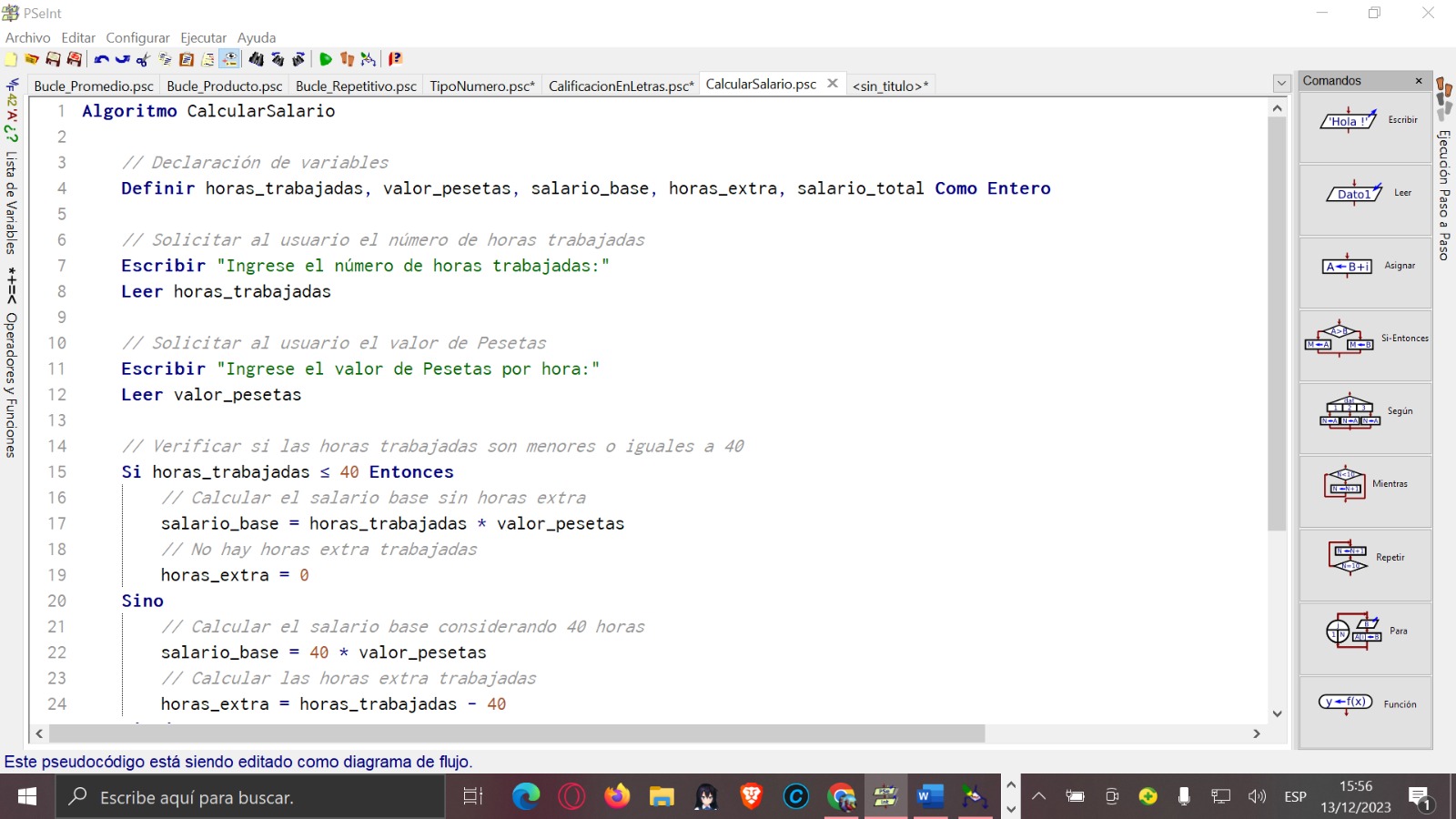
**Digrama de flujo:**

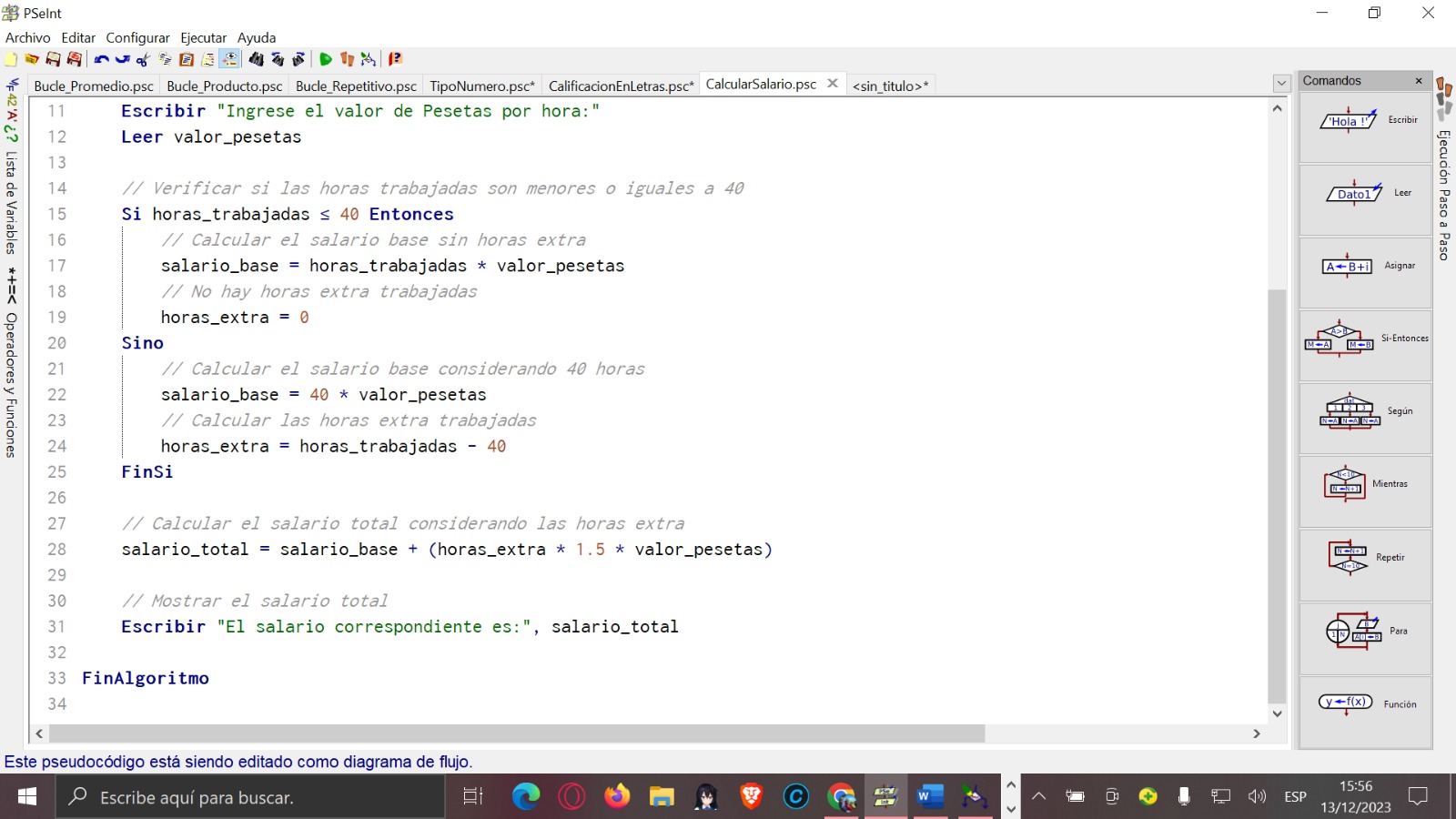


**Problema: Pago de salarios**

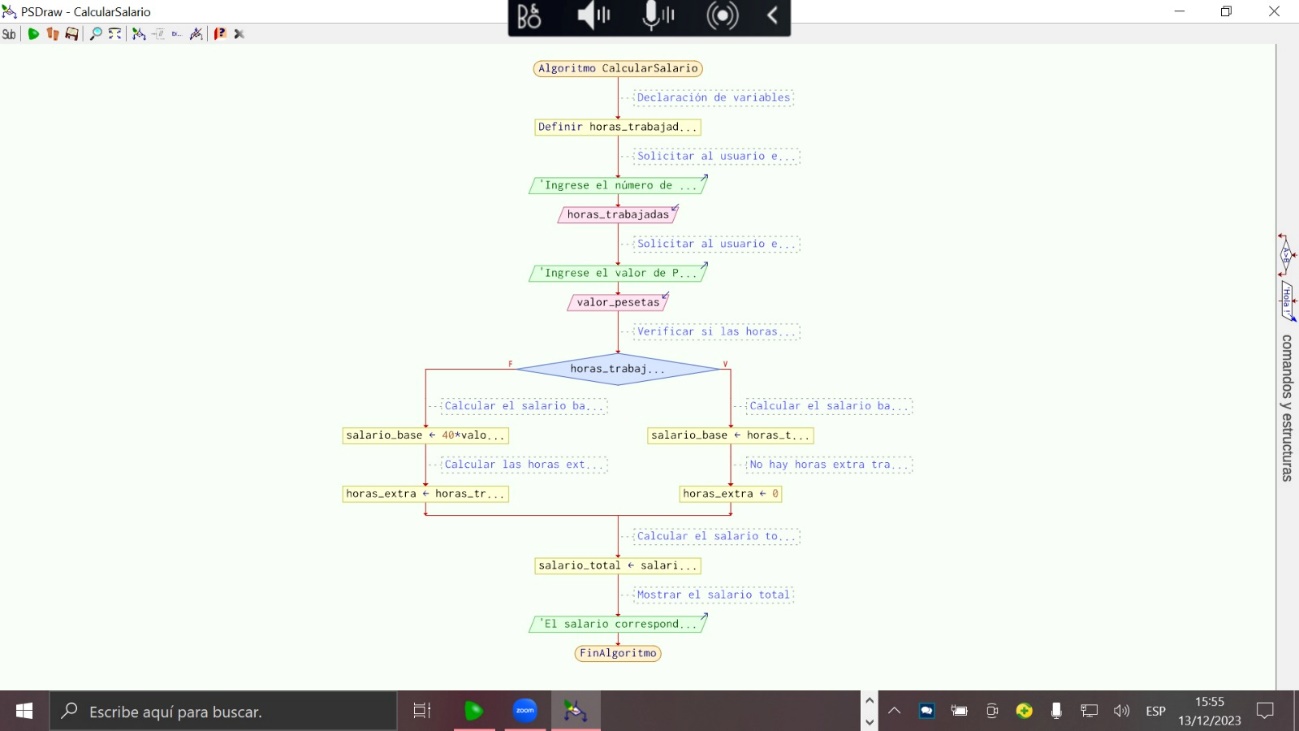
Una empresa paga a sus 100 operarios semanalmente, de acuerdo con el número de horas trabajadas, a razón de P pesetas por hora y de 1.5 pesetas por cada hora extra. Las horas extras son las que exceden de 40h. Hacer el organigrama que dado el número de horas trabajadas y el valor de P calcule el salario que le corresponde.

**Implementacion**:



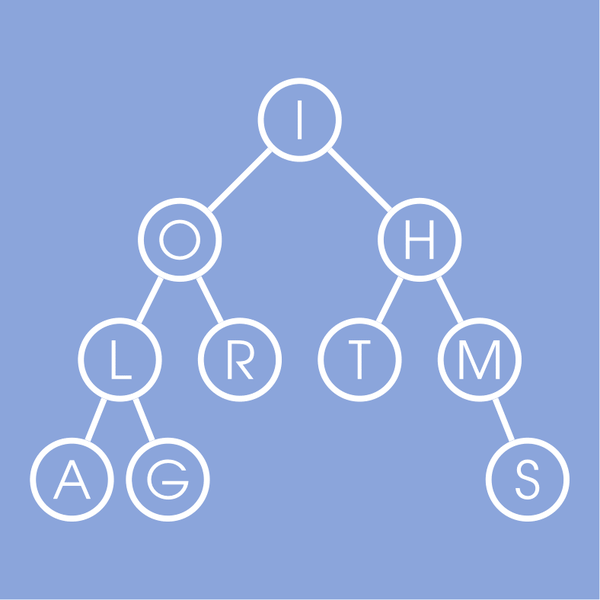


**Digrama de flujo:**



**Tipos de datos, estructuras de datos y tipos de datos abstractos**

En el mundo de la programación, la manipulación y representación de datos son aspectos cruciales que definen la eficiencia y funcionalidad de un software. En este contexto, nos adentramos en la comprensión profunda de tres conceptos fundamentales: tipo de datos (TD), estructura de datos (ED) y tipo de datos abstracto (TDA). Aunque estos términos comparten el objetivo de gestionar información, suelen generar confusión debido a sus matices específicos.



**Tipo de Datos (TD):** Se refiere al conjunto de valores que una variable puede tomar en un lenguaje de programación y las operaciones que pueden realizarse con dicha variable. Se ejemplifica con el tipo boolean en Pascal y se destaca la importancia de comprender el conjunto de valores y operaciones asociadas.

**Estructura de Datos (ED):** Consiste en una colección de variables organizadas de manera específica. La célula se presenta como la unidad básica de una estructura de datos. Se menciona la posibilidad de enlazar células usando apuntadores, que son valores que dirigen a una célula específica. Estos apuntadores pueden implementarse como cursores o punteros, según el contexto.

**Tipo de Datos Abstracto (TDA**): Se define por un modelo matemático y un conjunto de operaciones sobre ese modelo. Se distingue entre la definición y la implementación de un TDA. La definición implica establecer el modelo y las operaciones de manera clara y sin ambigüedad. La implementación implica encontrar una estructura de datos adecuada y escribir procedimientos que cumplan con las operaciones propuestas.

**Definición del TDA:** Se presenta un ejemplo con el TDA Número Complejo, especificando modelo (par ordenado de números reales) y conjunto de operaciones (suma, diferencia, producto, módulo, argumento, leer número, imprimir número) con descripciones precisas de cada operación.

**Implementación del TDA:** Implica encontrar una estructura de datos adecuada y escribir procedimientos que ejecuten las operaciones definidas. Se destaca la importancia de la concordancia entre la definición y la implementación para garantizar el correcto funcionamiento del programa.

**Uso del TDA**: Se señala que, si el lenguaje de programación ya tiene un tipo estándar que coincide con la definición del TDA, puede utilizarse sin necesidad de una implementación especial.

**CUESTIONARIO**

1. Que tipo de expresion se utiliza para evaluar condiciones en un algoritmo?

Relacional

2. Como se representa asignacion de valor en pseudocodigo?

Signo "="

3. Que es un algoritmo?

Conjunto de instrucciones que se siguen para resolver un

problema.

4. Que tipo de operador se utiliza para combinar condiciones en un algoritmo?

Logico

5. Que tipo de dato de utiliza comunmente para almacenar valores booleanos en un

algoritmo?

Booleano

6. Cual de los siguientes operadores en un operador aritmetico?

Aritmeticos: + - \\* /

7. Que es el pseudocodigo?

Un conjunto de simbolos y convicciones que se utilizan para

representar algoritmos.

8. Cual de las siguientes estructuras de control permite ejecuttar un bloque de codigo

solo si se cumple una condicion?

Decisiva

9. Cual es el proposito principal del pseudocodigo en la programacion?

Para representar de forma mas ACCESIBLE algun algoritmo.

10. Cual esa una tecnica para mejorar la eficiencia de un algoritmo?

Reducir la cantidad de operaciones realizadas.

11. Cual es el resultado de la siguiente expresion logica?

(True AND False) OR (False AND True) False

12. Cual de las siguientes opciones NO es un operador relacional?

% | Operadores relacionales: == / >= / <= / <>

13. Cuales son los elementos basicos de un algoritmo?

Intrucciones, estructuras de control y datos de salida.

14. Cual es la estructura que permite ejecutar un bloque de codigo un numero

especifico de veces?

Repetitiva

15. Cuales son las ventajas de utilizar diagramas de flujo para representar

algoritmos?

Son mas faciles de entender y seguir

16. Por que es importante la precision en los algoritmos?

Los algoritmos precisos

garantizan la solucion correcta del problema.

17. Cual es el proposito principal de un arreglo en la programacion?

Almacenar multiples valores de una sola variable. (del mismo tipo)

18. Cual es el proposito de las estructuras secuenciales?

Ejecutar el codigo en un

orde especifico

19. Cual es el proposito de las variables en un algoritmo?

Almacenar valores

intermedios y resultados.

20. Que herramientas computacionales existen para representar algoritmos?

Programas

para crear diagramas de flujo y editores de pseudocodigo.

21. Que es variable de control en un bucle?

Variable que controla la ejecucion del bucle.

22. Cuando usar el \_do-while\_?

Cuando un bloque de codigo debe ejecutarse al menos una vez y luego un numero especifico de veces mas.

23. Cual es el proposito de un bucle enn programacion?

Repetir codigo varias veces

24. Como se evita un bucle infinito?

Utilizando una condicion de salida que se cumpla en algun momento para cerrar el buble.

25. Cual es el proposito de la palabra 'continue' en un bucle?

Para saltar el resto del codigo en el bucle y continuar con la siguiente iteracion.

26. Que hace el bucle 'while'?

Ejecuta el bloque de codigo solo si se cumple la condicion establecida.

27. Cual es el contador en el contexto de algoritmos repetitivos?

Es una variable que lleva la cuenta de las repeticiones

28. Por que es importante la condicion de salida en un bucle?

Para evitar que se ejecute indefinidamente.

29. Cual es proposito de la palabra 'break' en un bucle?

Salir del bucle de manera anticipada

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | |
|  | Bibliografía  [Algoritmos y estructuras de datos - https://concepto.de](https://concepto.de/computacion/#:~:text=La%20computaci%C3%B3n%20es%20la%20ciencia,Estructura%20de%20datos%20y%20Algoritmos).  [Definicion de algoritmos - https://conceptodefinicion.de/computacion/](https://conceptodefinicion.de/computacion/)  [Informatica - https://es.wikipedia.org](https://es.wikipedia.org/wiki/Inform%C3%A1tica)  [Introduccion a la computacion - https://es.wikiversity.org](https://es.wikiversity.org/wiki/Introducci%C3%B3n_a_la_computaci%C3%B3n)  [Definicion de computacion - https://definicion.de/computacion/](https://definicion.de/computacion/)  [Tipos de datos de SQL - IBM](https://www.ibm.com/docs/es/iis/11.5?topic=stage-sql-data-types%7b)  [Constante & variables - https://www.mikroe.com](https://www.mikroe.com/ebooks/microcontroladores-pic-programacion-en-c-con-ejemplos/variables-y-constantes%23:~:text=Las%20variables%20se%20almacenan%20en,bytes)%20depende%20de%20su%20tipo.&text=Una%20constante%20tiene%20las%20mismas,durante%20la%20ejecuci%C3%B3n%20de%20programa).  <Que-es-un-diagrama-de-flujo> - <https://www.lucidchart.com>  [Tipos de datos - https://www.tecnologias-informacion.com](https://www.tecnologias-informacion.com/tipos-sql.html%7b)  [Identificadores en la programacion con C++ - https://www2.eii.uva.es{](https://www2.eii.uva.es/fund_inf/cpp/temas/2_tipos_variables/identificadores.html)  [Constantes y variables - https://www.ibm.com](https://www.ibm.com/docs/es/tcamfma/6.3.0?topic=tesl-constants-variables-4%7b)  [Operadores y expresiones - https://www.luisllamas.es](https://www.luisllamas.es/programacion-operadores-y-expresiones/)  [Abstraccion de datos - https://www.cursosdesarrolloweb.es](https://www.cursosdesarrolloweb.es/blog/abstraccion-programacion-orientada-objetos)  [Estructuras Secuenciales - https://www.ibm.com](https://www.ibm.com/docs/es/integration-bus/10.0?topic=dfdl-understanding-logical-structure/es/integration-bus/10.0?topic=dfdl-understanding-logical-structure)  [Construcción de Algoritmos - https://diegotch.webcindario.com](https://diegotch.webcindario.com/EstructurasCondicionales.pdf).  [Diagramas de Flujo - https://disenowebakus.net](https://disenowebakus.net/operadores-logicos.php)  [Estrutcuras repetitivas - https://uapa.cuaieed.unam.mx](https://uapa.cuaieed.unam.mx/sites/default/files/minisite/static/b62c06c7-6c36-48e9-9b56-9f164c1aaa76/UAPA-estructuras-repeticion/index.html) | |  |
|  | |