



# Argentina Programa





# Clase 04: Primeros Algoritmos

## Introducción Lógica Computacional



# Agenda

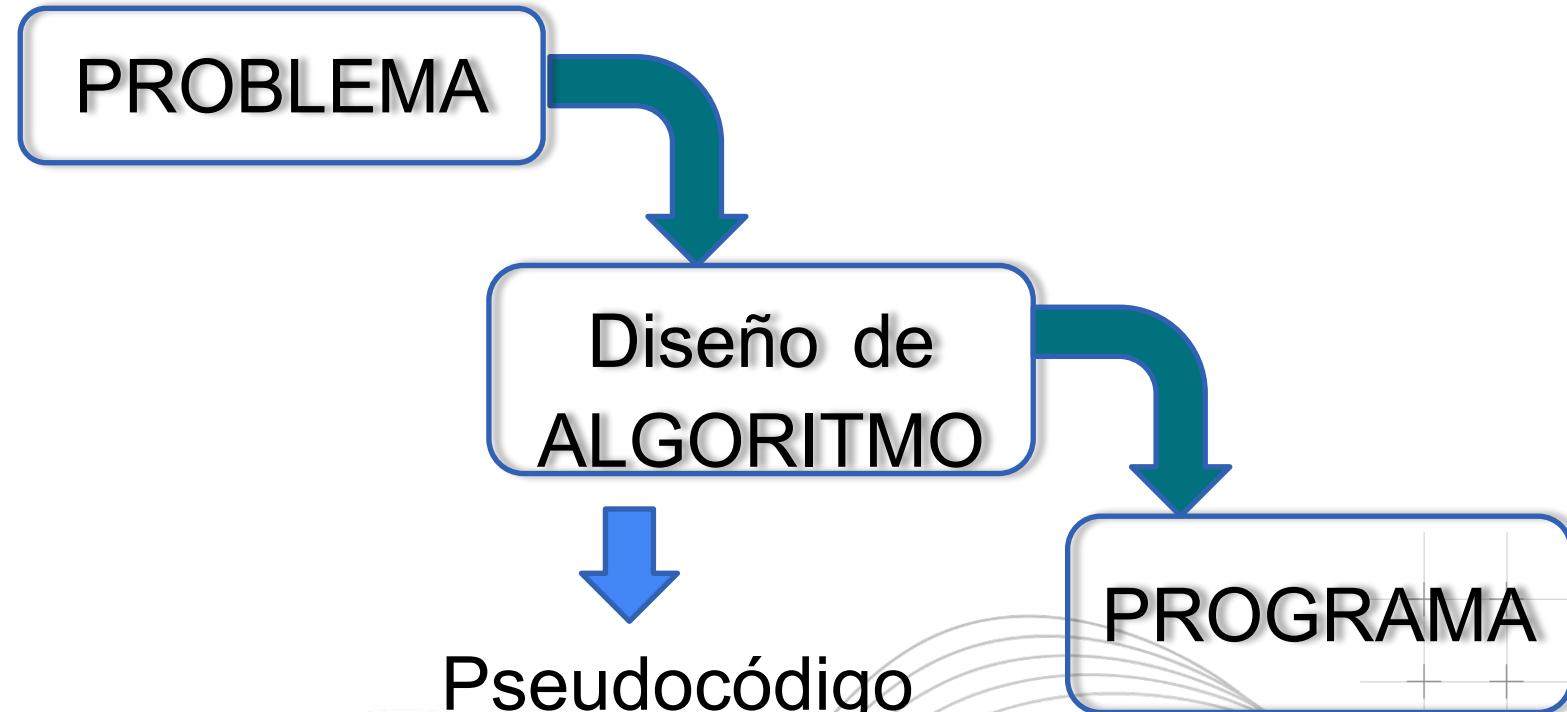


Desafíate a crear, diseñar y elaborar algoritmos  
que resuelvan problemas de la vida real

- Problema – Algoritmo - Programa
  - Crear Algoritmos utilizando pseudocódigo
- Ejercitación



# Problema – Algoritmo - Programa





## Fases en la resolución de problemas

- **Análisis del problema:** debe definirse cuál es el resultado o la solución esperada, los datos de entrada y qué debe hacer el programa para llegar a la solución.
- **Diseño del algoritmo:** se diseña el algoritmo que permitirá resolver el problema.
- **Codificación:** se escribe el programa a partir del algoritmo diseñado previamente, utilizando un lenguaje de programación. El diseño del algoritmo es independiente del lenguaje de programación que se utilice para implementar el programa.
- **Compilación y ejecución:** el programa fuente debe ser traducido a lenguaje máquina para la posterior ejecución de cada una de las instrucciones por parte del procesador de la computadora.
- **Verificación y depuración:** se comprueba que el programa obtenga los resultados esperados con una amplia variedad de datos de entrada. La depuración consiste en identificar y corregir los posibles errores que puedan encontrarse.
- **Documentación:** deben describirse los pasos del proceso de resolución del problema e incluir diagramas de flujo, manuales de usuario, definición de requerimientos, etcétera.
- **Mantenimiento:** el programa debe actualizarse y modificarse siempre que sea necesario, para cumplir con las necesidades de cambios de los usuarios. Para facilitar el proceso de mantenimiento, es muy importante contar con documentación clara y precisa.





## Diseño de algoritmos

- **Divide y vencerás:** Esta técnica implica dividir el problema en partes más pequeñas, ya que cada una de estas partes es más fácil de resolver.
- **Diseño descendente:** A medida que se avanza en la descomposición del problema original en subproblemas más pequeños, se amplían los detalles de la descripción del algoritmo que permite resolver el problema original.

Existen más técnicas pero estas son las más eficientes y utilizadas. Estas técnicas no son mutuamente exclusivas, y a menudo se combinan para resolver problemas complejos.

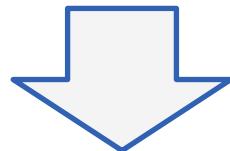




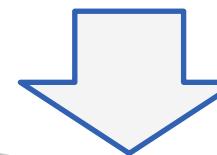
Diseño de  
ALGORITMO

¿Cómo creo un algoritmo?

**Secuencia Finita, Repetible y  
Correcta de pasos. (instrucciones)**



Pseudocódigo



Diagramas de flujo





## ¿Qué es el pseudocódigo?

El pseudocódigo se asemeja al lenguaje de programación real en términos de su estructura básica y uso de construcciones como bucles, condicionales y funciones, pero utiliza una sintaxis más flexible y menos rigurosa.

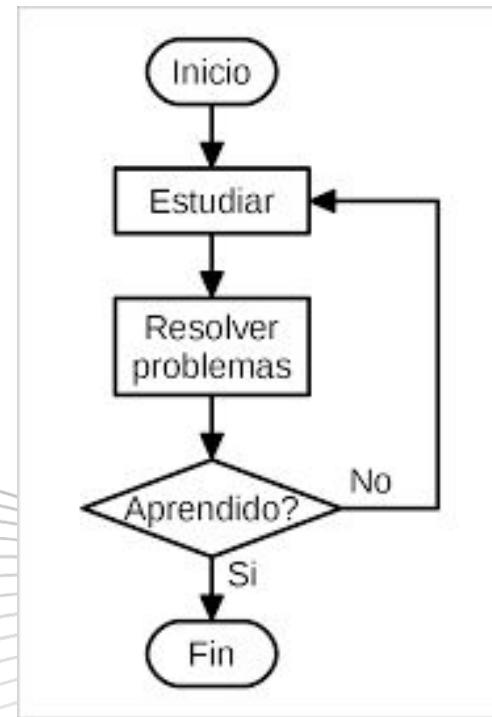
```
Inicio
    aprobados ← 0
    reprobados ← 0
    resultado ← 0
    estudiantes ← 1
    Mientras estudiantes <= 10
        Leer resultado
        Si resultado == 1 entonces
            aprobados ← aprobados + 1
        SiNo
            reprobados ← reprobados + 1
        FinSi
        estudiantes ← estudiantes + 1
    FinMientras
    Mostrar aprobados
    Mostrar reprobados
    Si aprobados > 8 entonces
        Mostrar "Aumentar la colegiatura"
    FinSi
Fin
```





## ¿Qué es un diagrama de flujo?

Un diagrama de flujo es una forma de representar gráficamente un algoritmo. Estos diagramas utilizan una serie de símbolos con significados especiales y son la representación gráfica de los pasos de un proceso. Estos símbolos están conectados mediante el uso de flechas que indican la secuencia de los pasos del algoritmo. Para evitar ambigüedades en la interpretación de los símbolos, estos han sido estandarizados mediante normas ANSI (American National Standards Institute).





Pasos para crear un algoritmo:

1. Determinar cuál es el objetivo
2. Identificar Entradas y Salidas
3. Describir los pasos (instrucciones)





## 1. Determinar cuál es el objetivo

Si no está claro el objetivo, entradas y salidas, no es posible desarrollar el algoritmo que resuelva el problema.





## 2. Identificar las entradas

**¿Cuáles son las entradas?**: qué información es conocida y se la puedo pedir al usuario

Se determina el estado inicial





## 3. Identificar las salidas

**¿Cuáles son las salidas?**: Qué información tiene que mostrar al usuario y cuándo.

Se determina el estado final





## 4. Describimos el proceso

Detallamos todos los pasos a realizar

- Utilizamos verbos en infinitivo
- Cada oración realiza una sola acción o realiza una toma de decisión.





## Expresiones y operadores

Una **expresión** es la combinación de variables, constantes, operadores, funciones y reglas. Se clasifican en:

- Aritméticas
- Relacionales
- Lógicas
- De carácter

Un **operador** es un símbolo especial que indica que se debe efectuar una determinada operación.





## ¿Qué son los datos?

Los datos son valores que se utilizan para realizar operaciones y procesos en un programa. El tipo de dato define el espacio que se utiliza en memoria para almacenar los valores.

- Numéricos (enteros, reales)
- Lógicos (verdadero, falso)
- Carácter y cadena
- De carácter

Un **operador** es un símbolo especial que indica que se debe efectuar una determinada operación.





## Variables y Constantes

Una **variable** es un espacio reservado en memoria cuyo contenido (valor) puede cambiar durante la ejecución del programa. Esta tiene asociada un identificador o nombre, un tipo de dato y un valor, y solo puede tomar valores permitidos según su tipo de dato.

Las **constantes** representan objetos cuyos valores no pueden cambiar durante la ejecución del programa.



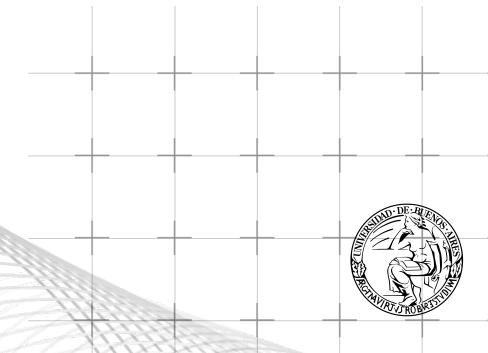


## ¿Qué son las palabras reservadas?

Las palabras reservadas poseen un significado especial para el lenguaje y no pueden ser utilizadas como identificadores.

Algunos ejemplos de palabras reservadas en pseudocódigo pueden ser:

- inicio
- fin
- entero
- real
- mientras
- si
- sino
- entonces
- para
- según
- sea
- repetir
- función
- lógico
- carácter
- cadena





## EJERCICIOS EN PSEUDOCÓDIGO

Existen muchas formas y reglas distintas para representar la estructura de un algoritmo en pseudocódigo.

Nosotros seguiremos la siguiente estructura:

- **Cabecera del algoritmo.**
- **Declaración de estructuras de datos (arreglos y registros).**
- **Declaración de variables y constantes.**
- **Sentencias ejecutables.**

*algoritmo <nombre de algoritmo>  
tipo  
    <definición de arreglos y registros>  
var  
    <tipo de dato>: <nombre de variable, nombre de variable, etc.>  
    <tipo de dato>: <nombre de variable>*

*const  
    <tipo de dato>: <nombre de constante> ← <valor>  
    <tipo de dato>: <nombre de constante> ← <valor>*

*inicio  
    <sentencia ejecutable 1>  
    <sentencia ejecutable 2>*

*< sentencia ejecutable n>*

*fin*





## EJERCICIOS EN PSEUDOCÓDIGO

**EJEMPLO:** Implementar un algoritmo que permita sumar dos números enteros ingresados por teclado y muestre el resultado por pantalla.

```
algoritmo suma_dos_numeros
var
    entero: a, b, suma
inicio
    leer(a, b)
    suma ← a + b
    mostrar(suma)
fin
```

```
Algoritmo Suma
Definir A, B, C Como Entero

Escribir "Ingrese el primer numero:"
Leer A

Escribir "Ingrese el segundo numero:"
Leer B

C ← A+B

Escribir "El resultado es: ",C

FinAlgoritmo
```





## EJERCICIOS EN PSEUDOCÓDIGO

1. Escribir un algoritmo en lenguaje natural para: “Preparar un Mate”
2. Descargar PsEInt
3. Escribir un algoritmo que permita multiplicar dos números enteros ingresados por teclado y muestre el resultado por pantalla.





## ESTRUCTURAS SECUENCIALES

Las estructuras secuenciales son aquellas en las que las acciones se ejecutan una a continuación de la otra, en secuencia. Por ejemplo:

```
leer(A)  
leer(B)  
suma ← A + B  
mostrar(suma)
```





## ESTRUCTURAS SECUENCIALES

Las estructuras secuenciales son aquellas en las que las acciones se ejecutan una a continuación de la otra, en secuencia. Por ejemplo:

```
leer(A)  
leer(B)  
suma ← A + B  
mostrar(suma)
```





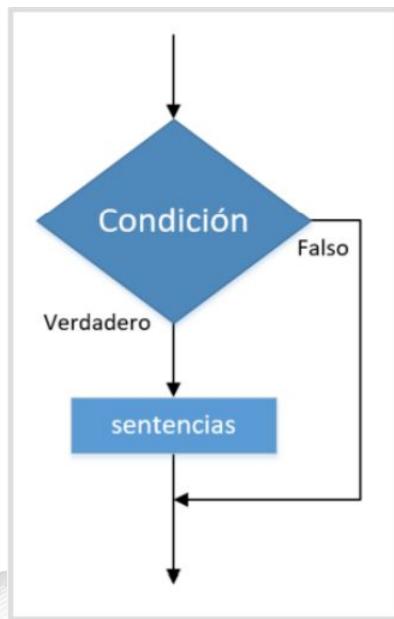
## Estructuras condicionales

Las estructuras de control condicionales son utilizadas para ejecutar un conjunto de sentencias si se cumple una condición. Esto permite modificar el orden de ejecución de las sentencias según el resultado de una expresión que se evalúa. El tipo de dato del resultado de la condición debe ser de tipo lógico.

Las estructuras condicionales se clasifican en: simples, dobles y múltiples

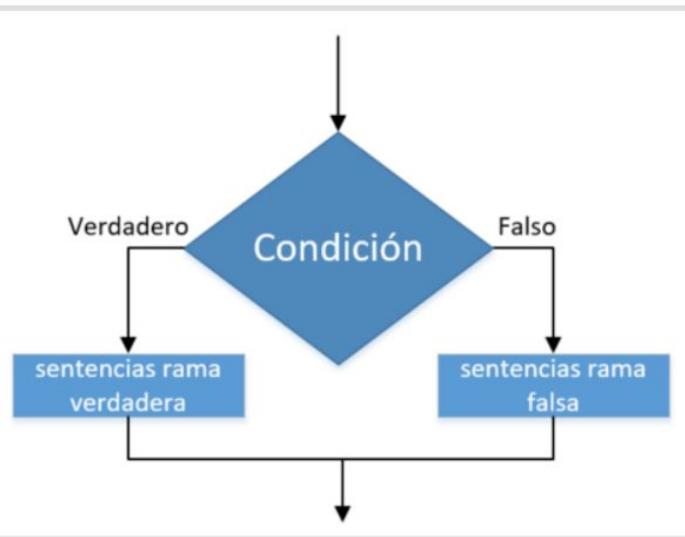


## Estructuras condicionales simples



```
Algoritmo condicional_Simple
Escribir ("Ingresa un número:")
Leer num
Si num > 0 Entonces
    Escribir "El número es positivo"
Fin Si
FinAlgoritmo
```

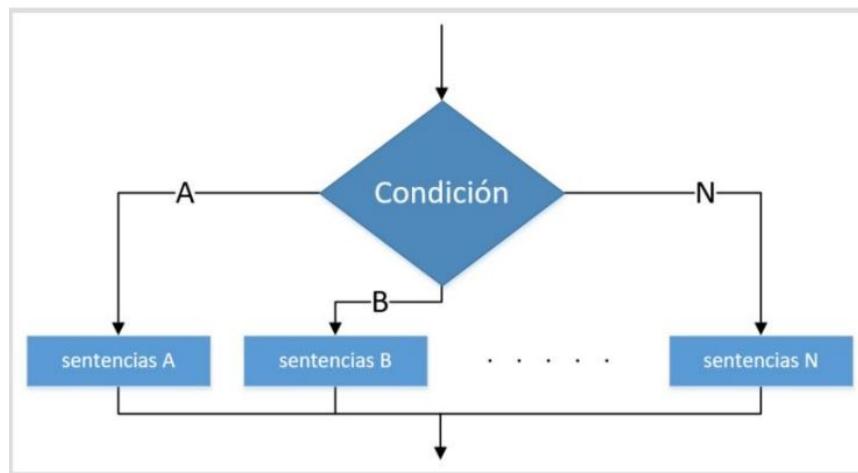
## Estructuras condicionales dobles



```
1 Algoritmo condicional_Simple
2 Escribir ("Ingresa un número:")
3 Leer num
4 Si num > 0 Entonces
5   Escribir "El número es positivo"
6 SiNo
7   Escribir "El número es negativo"
8 Fin Si
9 FinAlgoritmo
```



## Estructuras condicionales múltiples



```
1 Algoritmo condicional_múltiple
2 Escribir ("Ingrese un número entero comprendido entre 1 y 3 :")
3 Leer num
4 Segun num Hacer
5   1: Mostrar ("Viernes")
6   2: Mostrar ("Sabado")
7   3:
8   0: Mostrar ("Domingo")
9 De Otro Modo:
10  Mostrar ("Opción no válida")
11 Fin Segun
12 FinAlgoritmo
```



## EJERCICIOS CON CONDICIONALES

1. Implementar un algoritmo que permita ingresar tres números enteros por teclado y que determine cuál es el mayor de ellos.
2. Implementar un algoritmo que permita ingresar dos números enteros por teclado. Luego, el usuario debe elegir si desea mostrar la suma o multiplicación de ambos números





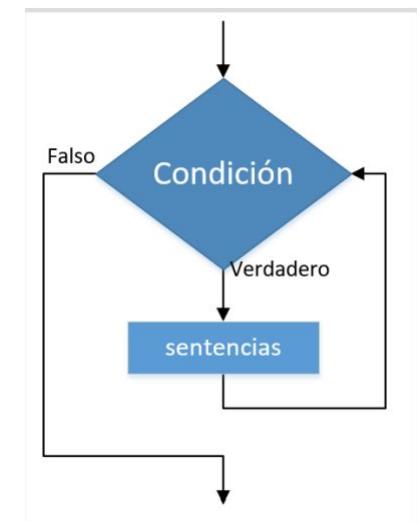
## Estructuras repetitivas

Las instrucciones iterativas o de repetición permiten ejecutar una secuencia de instrucciones más de una vez. Un bucle es una sección de código que se repite. Es decir que cuando se termina de ejecutar la última instrucción del conjunto, el flujo de control retorna a la primera sentencia y comienza una nueva repetición. Se denomina iteración al hecho de repetir la ejecución de una secuencia de acciones.



## Estructura mientras

En la estructura mientras, se evalúa primero la condición y si esta es verdadera, entonces se ejecutan las sentencias definidas en el bucle. Al finalizar la iteración, se vuelve a evaluar la condición para volver a ejecutar una nueva iteración hasta que la condición dé como resultado un valor falso.





## Estructura mientras

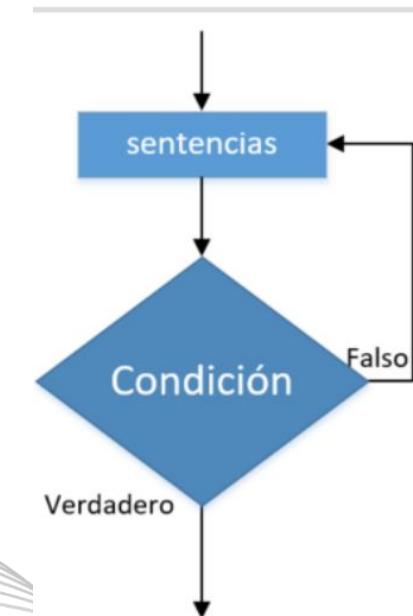
```
3 Algoritmo sin_titulo
4     // Declarar las variables
5     Definir numero1, numero2, i, producto Como Entero
6     // Solicitar al usuario que ingrese los números enteros
7     escribir("Ingrese el primer número entero positivo: ")
8     leer numero1
9     escribir("Ingrese el segundo número entero positivo: ")
10    leer numero2
11    // Inicializar la variable producto en cero
12    producto = 0
13
14    // Calcular el producto utilizando el algoritmo de sumas sucesivas con la estructura mientras
15    i = 1
16    mientras i ≤ numero2 hacer
17        producto = producto + numero1
18        i = i + 1
19    fin mientras
20
21    // Mostrar el producto por pantalla
22    escribir("El producto de "), numero1, (" y "), numero2, (" es: "), producto
23 FinAlgoritmo
```





## Estructura repetir-hasta

Esta estructura repetitiva se utiliza cuando se desea que se ejecute una iteración al menos una vez antes de comprobar la condición de repetición. Es decir, primero se ejecuta el bucle y luego se comprueba la condición para reproducir una nueva iteración o no del bucle. Esta estructura ejecuta el bucle de repetición mientras el resultado de la condición que se evalúa sea falso.





## Estructura repetir/ hasta

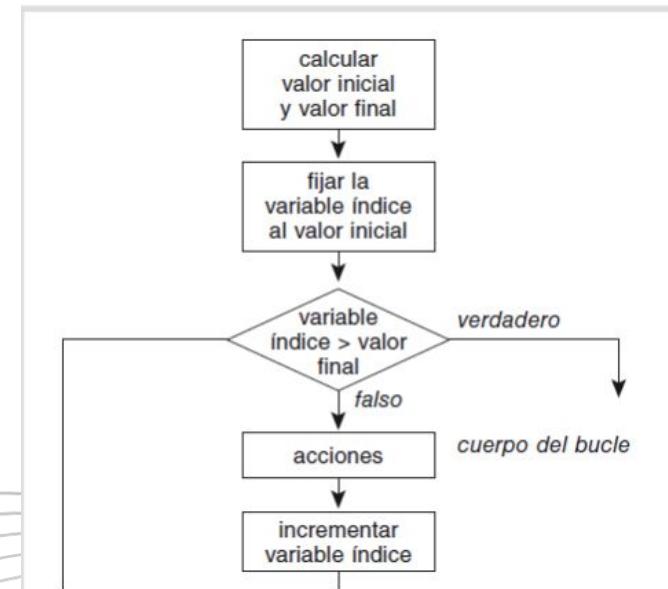
```
2
3 Algoritmo sin_titulo
4     // Declarar las variables
5     | Definir i, num, suma como entero
6     | Definir promedio como real
7
8     // Inicializar las variables
9     suma = 0
0
1     // Solicitar al usuario que ingrese los 10 números enteros
2     para i = 1 hasta 4 hacer
3         escribir("Ingrese el número "), i, (: )
4         leer num
5         // Acumular los números ingresados en la variable suma
6         suma = suma + num
7     fin para
8     // Calcular el promedio
9     promedio = suma / 4
0     // Mostrar el promedio por pantalla
1     escribir("El promedio de los 4 números ingresados es: ") , promedio
2 FinAlgoritmo
3
```



## Estructura para/desde

En muchas ocasiones se conoce el número de veces que se desea que un bucle se repita.

Para ejecutar el bucle, se evalúa una condición que debe ser verdadera. La condición más comúnmente usada es que el valor del índice sea menor o igual a un valor final. Si esto se cumple, entonces se ejecuta el bucle. Al finalizar, se incrementa o decrementa el índice y se vuelve a evaluar la condición. Cuando la condición es falsa, se ignora el bucle y se continúa con el resto del algoritmo.





## Estructura para/desde

```
> Algoritmo estructura_Para_Desde
|   acumulado = 0
|   para i = 1 hasta 10 hacer
|       acumulado = acumulado + i
|   fin para
|   Escribir("La suma de los números del 1 al 10 es: "), acumulado
> FinAlgoritmo
)
```



## EJERCICIOS CON ESTRUCTURAS REPETITIVAS

1. Realizar un programa que solicite al usuario que ingrese una clave, y que continúe pidiéndole la clave hasta que ingrese la clave correcta (por ejemplo, "1234").
2. Hacer un algoritmo en Pseint para calcular la suma de los primeros cien números con un ciclo mientras.
3. Realizar un programa que pida al usuario que ingrese un número entero positivo, y que muestre por pantalla la suma de los números enteros pares entre 1 y el número ingresado por el usuario.



# CONSULTAS?

Muchas Gracias!

