PROGRAMACIÓN I

1º AÑO

CLASE Nº 2

Algoritmos de Programación

Contenido

En la clase de hoy trabajaremos los siguientes temas:

- Definición y características de los algoritmos.
- Notación para diseño de algoritmos: diagramas de flujo y pseudocódigo.
- Estructuras algorítmicas: secuencial, de decisión y cíclica.
- Elementos de un programa: instrucciones de entrada y salida, comentarios, identificadores, variables, tipos de datos y expresiones.
- Prueba de escritorio.

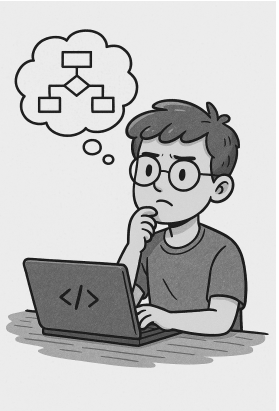
1. Presentación

¡Bienvenidos a la segunda clase de Programación 1! En la clase pasada trabajamos sobre la introducción a la programación, y hoy nos centraremos en los algoritmos, su estructura y representación. Comprender estos conceptos nos permitirá estructurar código de manera eficiente y preparar la base para lenguajes de programación.

Pregunta para reflexión

Antes de comenzar, piensen en esto: Si tuvieran que explicarle a una computadora cómo atarse los cordones de los zapatos,

¿cómo lo harían?



(Tomémonos unos minutos para pensar)

Lo que acaban de hacer es **descomponer** un problema en una secuencia de pasos lógicos. Eso, en esencia, es un algoritmo. Hoy vamos a aprender cómo los algoritmos son fundamentales en programación, cómo los usamos en nuestro día a día y cómo se estructuran.

2. Desarrollo

2.1. Definición de Algoritmo

Según la RAE: *conjunto ordenado y finito de operaciones que permite hallar la solución de un problema.*

Los algoritmos, como indica su definición oficial, son una serie de pasos que permiten obtener la solución a un problema.

El lenguaje algorítmico es aquel que implementa una solución teórica a un problema indicando las operaciones a realizar y el orden en el que se deben efectuar.

Actividad inicial

Seguimos reflexionando: *¿En qué situaciones cotidianas crees que aplicamos algoritmos?*

Los algoritmos están presentes en numerosos aspectos de nuestra vida diaria, aunque a menudo no seamos conscientes de ellos. Desde tareas simples hasta procesos más complejos, seguimos una serie de pasos para alcanzar un resultado. PD: Piensa en actividades como seguir una receta de cocina, elegir la ruta más rápida para llegar a un destino, organizar tu agenda o incluso en la forma en que una aplicación recomienda contenido según tus preferencias.

2.2. Elementos de un algoritmo

- **Entrada:** Los datos iniciales que posee el algoritmo antes de ejecutarse.
- **Proceso:** Acciones que lleva a cabo el algoritmo.
- **Salida:** Datos que obtiene finalmente el algoritmo.

2.3. Fases en la creación de algoritmos

Hay tres fases en la elaboración de un algoritmo:

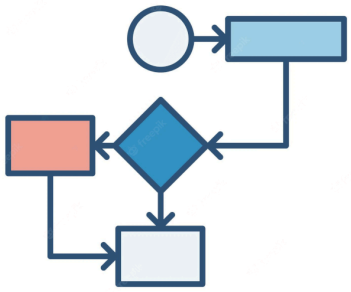
1. **Análisis:** En esta se determina cuál es exactamente el problema a resolver. Qué datos forman la entrada del algoritmo y cuáles deberán obtenerse como salida.
2. **Diseño:** Elaboración del algoritmo.
3. **Prueba:** Comprobación del resultado. Se observa si el algoritmo obtiene la salida esperada para todas las entradas.

2.4. Notaciones para diseño de algoritmos

Te invitamos a revisar este vídeo:

Conceptos básicos de Programación.

Diagramas de flujo



Es una representación gráfica de un proceso o algoritmo que utiliza símbolos estándar para mostrar los pasos secuenciales que se deben seguir.

Cada paso del proceso está representado por un símbolo específico, y las flechas indican el flujo o la dirección del proceso. Los diagramas de flujo son útiles para visualizar la lógica de un algoritmo, lo que facilita la comprensión, la comunicación y la planificación de soluciones.

Algunos símbolos comunes en los diagramas de flujo son:

- **Óvalo**: Representa el inicio o el fin del proceso.
- **Rectángulo**: Representa una acción o un paso del proceso.
- **Rombo**: Representa una decisión o una condición (sí/no).
- **Flechas**: Indican la dirección del flujo del proceso.

```
Algoritmo PrecioIndividual
    Escribir "Ingrese precio individual: "
    Leer precio
    Escribir "In
    Leer cantidac
    Escribir "In
    total_sin_iva
    total_con_iva
    Escribir "To
    Escribir "To
FinAlgoritmo
```

```
PSeInt - Ejecutando proceso PRECIOINDIVIDUAL
*** Ejecución Iniciada. ***
Ingrese precio individual:
> 100
Ingrese cantidad a comprar:
> 1
Ingrese calificación de parcial:
Total sin iva: 100
Total con iva: 112.7
*** Ejecución Finalizada. ***
```

Pseudocódigo

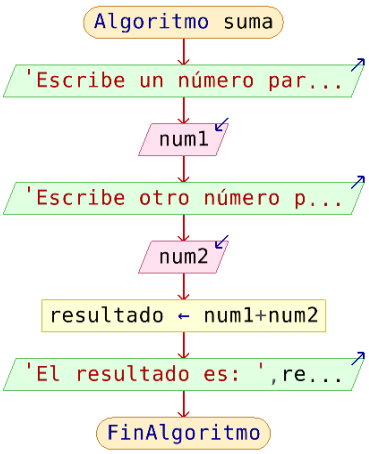
Es una forma de escribir un algoritmo de manera que sea fácilmente comprensible por los humanos, pero utilizando una estructura que imita el lenguaje de programación.

No sigue la sintaxis exacta de un lenguaje de programación, pero describe la lógica y los pasos del algoritmo de forma estructurada.

El pseudocódigo es útil para planificar y documentar algoritmos antes de implementarlos en un lenguaje de programación real.

Comparativa de herramientas

Diagrama de flujo



Pseudocódigo

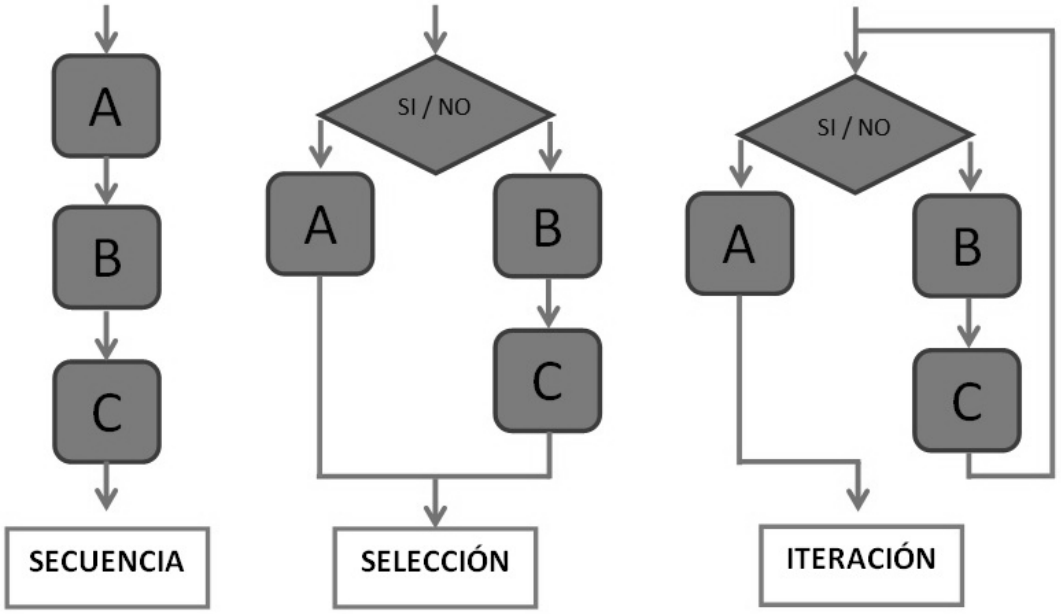
```
1  Algoritmo suma
2      Definir num1, num2, resultado Como Entero
3
4      Escribir "Escribe un número par:"
5      Leer num1
6
7      Escribir "Escribe otro número par:"
8      Leer num2
9
10     resultado ← num1 + num2
11
12     Escribir "El resultado es: ", resultado
13 FinAlgoritmo
```

En la imagen se presenta un ejemplo práctico de un algoritmo de suma, ilustrado tanto en diagrama de flujo como en pseudocódigo. Esto permite apreciar cómo cada metodología expresa los mismos pasos lógicos, desde la entrada de datos hasta la salida del resultado, destacando la utilidad de ambas herramientas en la planificación y desarrollo de programas.

Fundamentos de la programación

Las bases de la programación estructurada fueron enunciadas por Niklaus Wirdth. Según este científico cualquier problema algorítmico podía resolverse con el uso de estos tres tipos de instrucciones:

- **Secuenciales:** Instrucciones que se ejecutan en orden normal. El flujo del programa ejecuta la instrucción y pasa a ejecutar la siguiente.
- **Alternativas:** Instrucciones en las que se evalúa una condición y dependiendo si el resultado es verdadero o no, el flujo del programa se dirigirá a una instrucción o a otra.
- **Iterativas:** Instrucciones que se repiten continuamente hasta que se cumple una determinada condición.

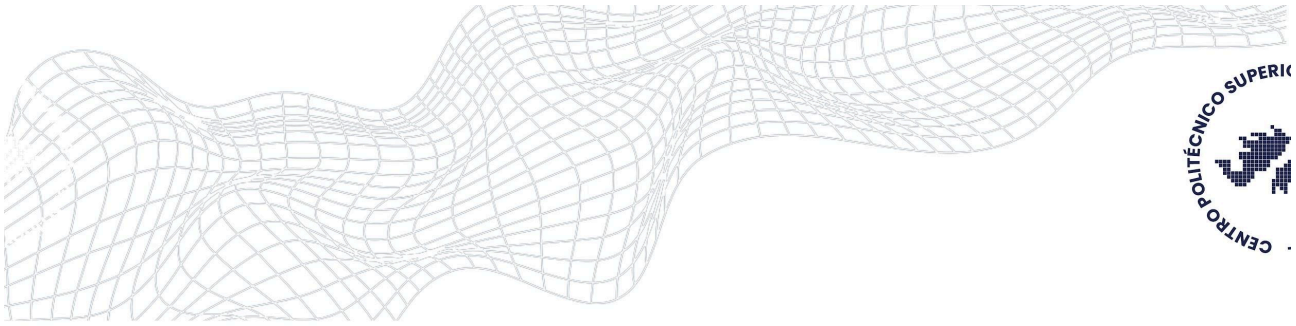


2.5 Creación de algoritmos

El diseño de un algoritmo consiste en definir las **acciones o instrucciones** necesarias para resolver un problema. Estas deben escribirse en **secuencia** y seguir reglas estrictas según el lenguaje de programación utilizado.

Tipos de Instrucciones

1. **Primitivas**: Operaciones básicas sobre los datos, como asignación y entrada/salida.
2. **Declaraciones**: Definen variables y subprogramas (obligatorias en pseudocódigo).
3. **Control**: Modifican el flujo de ejecución según condiciones, permitiendo decisiones y repeticiones.



Instrucciones Primitivas

Son acciones básicas sobre los datos del programa, como asignaciones y operaciones aritméticas.

Ejemplo de asignación en pseudocódigo:

```
a ← 10  // Se asigna el valor 10 a la variable 'a'
b ← a + 5  // 'b' toma el valor de 'a' más 5
```

*** Instrucción de Asignación**

Permite almacenar un valor en una variable usando el operador ←.

Ejemplo:

```
a ← 8   // 'a' ahora vale 8
b ← a   // 'b' toma el valor de 'a' (8)
```

Los valores pueden ser:

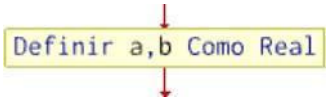
- Números (10, 3.5)
- Caracteres ('A', 'B')
- Textos ("Hola")
- Lógicos (Verdadero, Falso)
- Identificadores (asignando el valor de otra variable).

*** Operadores Algebraicos**

- + (suma), - (resta), * (multiplicación), / (división)
- mod (resto), div (división entera), ^ (potencia)
- La multiplicación y división tienen mayor prioridad que la suma y resta. Se pueden usar paréntesis para modificar la precedencia.

Instrucciones de Declaración

Permiten definir las variables que se usarán en el algoritmo. En pseudocódigo, se hace con la palabra *Definir*.



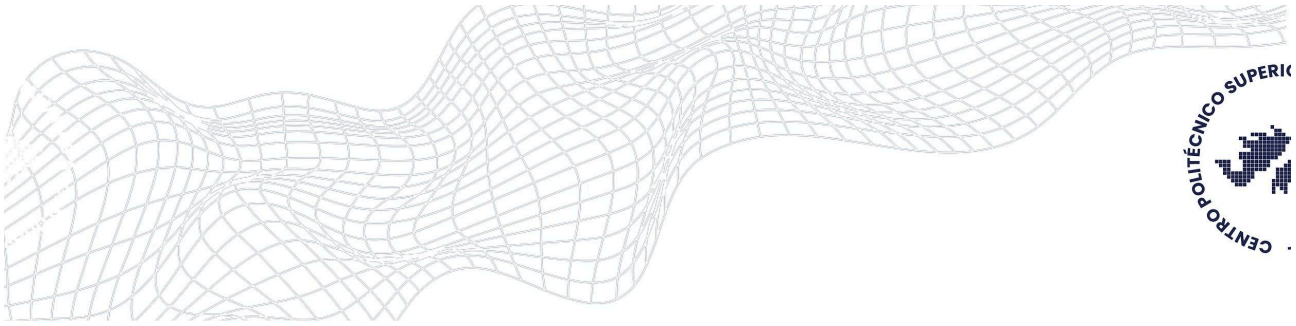
Ejemplo:

```
Definir nombre Como Cadena
Definir edad Como Entero
```

Estas instrucciones de declaración se utilizan para indicar el nombre y las características de las variables que se utilizan en el algoritmo.

Las variables son nombres a los que se les asigna un determinado valor y son la base de la programación.

Al nombre de las variables se le llama identificador.



Identificadores de variables

Los algoritmos utilizan **datos** que son almacenados en variables como habíamos mencionado, estas se identifican con un **nombre único** llamado identificador. Este debe:

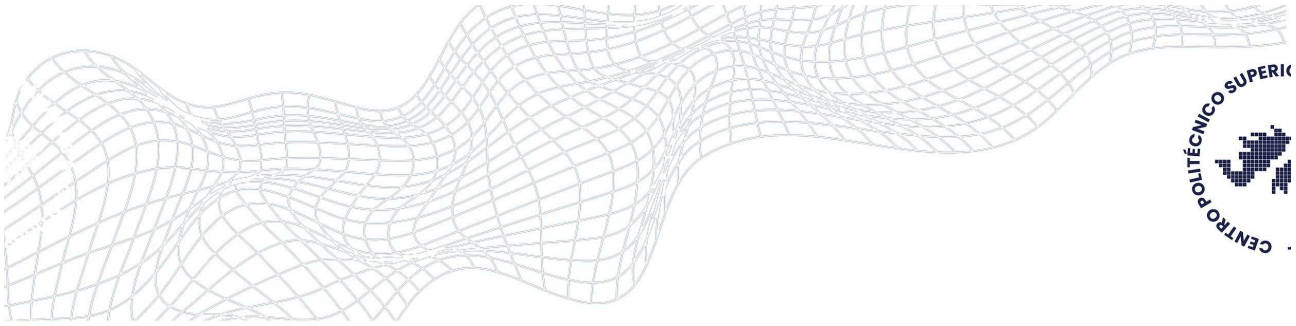
- Contener letras, números y el carácter _, comenzando siempre con una letra.
- No repetirse dentro del mismo algoritmo.
- Ser descriptivo (ejemplo: `saldoMensual` en lugar de `x`).
- Usar notación **Camel Case** para mayor legibilidad.

Tipos de Datos

- Entero: Números sin decimales.
- Real: Números con decimales.
- Carácter: Un solo símbolo alfanumérico.
- Texto: Conjunto de caracteres.
- Lógico (Booleano): Valores Verdadero o Falso.

Las constantes son valores fijos que no cambian durante la ejecución del algoritmo. Se representan en mayúsculas para diferenciarlas.

```
PI ← 3.1416
```



Instrucciones de Control

Permiten alterar el flujo de ejecución del algoritmo, dividiéndose en:

- **Condicionales (si, si-no, según)**
- **Bucles o iteraciones (mientras, repetir, para)**

2.6 Instrucciones de entrada y salida

*** Lectura de datos**

Es la instrucción que simula una lectura de datos desde el teclado. Se utiliza la sentencia ***Leer*** para recibir información del usuario.

```
Leer a
```

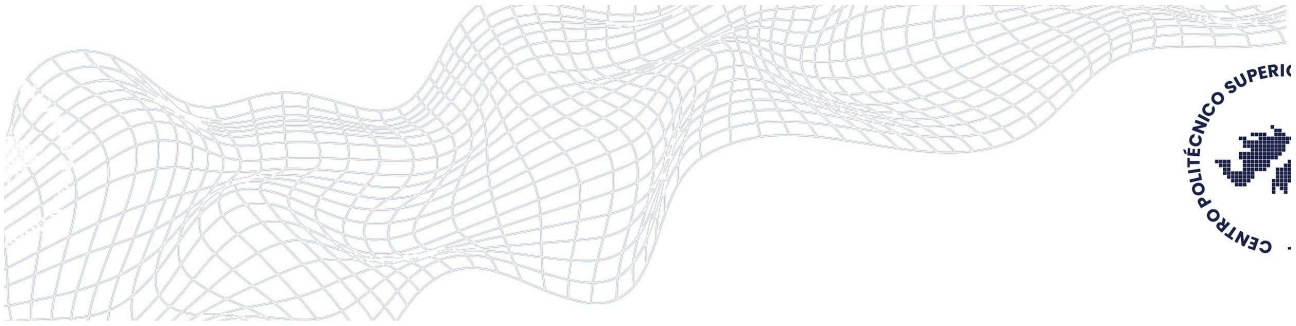
También podemos leer varias variables a la vez:

```
Leer a, b
```

*** Escritura de datos**

Funciona como la anterior pero usando la palabra Escribir. Simula la salida de datos del algoritmo por pantalla.

```
Escribir "El resultado es:", a
```



Ejemplo Completo

Algoritmo que multiplica dos números ingresados por el usuario:

```
Algoritmo Multiplicación
   Definir a, b, resultado Como Entero
   Escribir "Ingrese dos números:"
   Leer a, b
   resultado ← a * b
   Escribir "El resultado es:", resultado
FinAlgoritmo
```

2.7 Expresiones Lógicas

Todas las instrucciones de este apartado utilizan expresiones lógicas. Son expresiones que dan como resultado un valor lógico (verdadero o falso).

Son condiciones que devuelven **Verdadero** o **Falso**.

```
Si edad >= 18 Entonces
   Escribir "Mayor de edad"
FinSi
```

Lista de operadores relacionales

OPERADOR	SIGNIFICADO	EJEMPLO
>	Mayor que...	3 > 2
<	Menor que...	'ABC' < 'abc'
=	Igual que...	4 = 3
<=	Menor o igual que...	'a' <= 'b'
>=	Mayor o igual que...	4 >= 5
<>	Distinto que...	'a' <> 'b'

También se pueden combinar con operadores lógicos:

- AND (Y): Ambas condiciones deben cumplirse.
- OR (O): Basta con que una se cumpla.
- NOT (NO): Niega el valor de una condición.

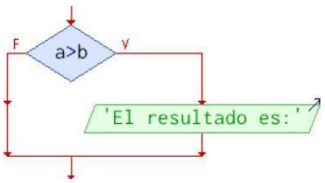
EXPRESIÓN	RESULTADO VERDADERO SI...
a > 8 Y b < 12	La variable a es mayor que 8 y (a la vez) la variable b es menor que 12.
a > 8 O b < 12	O la variable a es mayor que 8 o la variable b es menor que 12. Basta que se cumpla una de las dos expresiones.
a > 30 Y a < 45	La variable a está entre 31 y 44.
a < 30 O a > 45	La variable a no está entre 30 y 45.
NO a = 45	La variable a no vale 45.
a > 8 Y NO b < 7	La variable a es mayor que 8 y b es menor o igual que 7.
a > 45 Y a < 30	Nunca es verdadero.

2.8 Condicionales

Las condicionales se crean con la instrucción si (en inglés if). Esta instrucción evalúa una determinada expresión lógica y dependiendo de si

esa expresión es verdadera o no se ejecutan las instrucciones siguientes.

Esto es equivalente al siguiente diagrama de flujo:



Las instrucciones sólo se ejecutarán si la expresión evaluada es verdadera.

*** Alternativa Simple (`Si...Entonces`)**

Ejecuta un bloque de código si la condición se cumple.

```
Si edad >= 18 Entonces
   Escribir "Mayor de edad"
FinSi
```

*** Alternativa Doble (`Si...Entonces...SiNo`)**

Ejecuta un bloque si la condición es verdadera y otro si es falsa.

```
Si temperatura > 30 Entonces
   Escribir "Hace calor"
SiNo
   Escribir "El clima es agradable"
FinSi
```

*** Alternativa Múltiple (`Según...Hacer`)**

Útil cuando hay varias opciones posibles.

```
Según diaSemana Hacer
   Caso "Lunes": Escribir "Inicio de semana"
   Caso "Viernes": Escribir "Fin de semana"
   De Otro Modo: Escribir "Día normal"
FinSegún
```

2.9 Bucles o ciclos

Son estructuras de control que permiten ejecutar un bloque de código de manera repetitiva, mientras se cumpla una condición determinada. Son útiles cuando se necesita realizar una tarea varias veces sin tener que escribir el mismo código repetidamente.

*** Bucle Mientras (`Mientras...Hacer`)**

Ejecuta un bloque de código mientras la condición sea verdadera.

```
Mientras contador < 5 Hacer
   Escribir "Valor:", contador
   contador ← contador + 1
FinMientras
```

*** Bucle Repetir (`Repetir...Hasta Que`)**

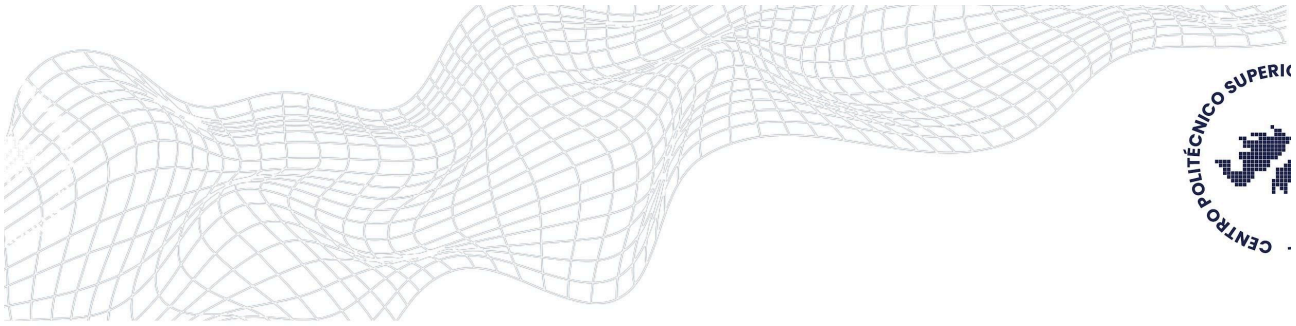
Ejecuta el bloque al menos una vez y se repite hasta que la condición sea verdadera.

```
Repetir
   Escribir "Introduce un número mayor a 10"
   Leer num
Hasta Que num > 10
```

*** Bucle Para (`Para...Hacer`)**

Ejecuta un número determinado de veces.

```
Para i ← 1 Hasta 5 Hacer
   Escribir "Número:", i
FinPara
```



*** Bucles Anidados**

Al igual que ocurría con las instrucciones si, también se puede insertar una estructura iterativa dentro de otra; pero en las mismas condiciones que la instrucción si. Cuando una estructura iterativa está dentro de otra se debe cerrar la iteración interior antes de cerrar la exterior (véase la instrucción si).

```
Para i ← 1 Hasta 3 Hacer
   Para j ← 1 Hasta 3 Hacer
      Escribir "Posición:", i, j
   FinPara
FinPara
```

2.10 Comentarios

En pseudocódigo los comentarios que se deseen poner (y esto es una práctica muy aconsejable) se ponen con los símbolos // al principio de la línea de comentario (en algunas notaciones se escribe **). Cada línea de comentario debe comenzar con esos símbolos:

```
Algoritmo NombreDelAlgoritmo
   instrucciones
   //comentario
   //comentario
   instrucciones
FinAlgoritmo
```

2.11 Prueba de Escritorio

Una prueba de escritorio es un tipo de prueba algorítmica, que consiste en la validación y verificación del algoritmo a través de la ejecución de las sentencias que lo componen (proceso) para determinar sus resultados (salida) a partir de un conjunto determinado de elementos (entrada).

*** Ejemplo de algoritmo**

Objetivo: Sumar dos números

Pasos del algoritmo:

1. Leer dos números, a y b.
2. Sumar los dos números.
3. Mostrar el resultado.

Entrada: a = 3, b = 5

Prueba de escritorio

1. **Inicializamos las variables**: `a = 3`, `b = 5`.
2. **Sumamos los números**: `resultado = a + b = 3 + 5 = 8`.
3. **Mostramos el resultado**: El resultado es 8.

Resumen

La prueba de escritorio muestra cómo se realiza cada paso del algoritmo:

- Se asignan los valores a las variables.
- Se realiza la operación (suma).
- Se obtiene el resultado final (8).

Actividad de cierre

Ahora vamos con una actividad de cierre, donde el objetivo es comprender el funcionamiento de un algoritmo sencillo en PSeInt mediante la realización de una prueba de escritorio y el análisis de distintos valores de entrada con la ayuda de la IA.

PASO 1 ›› Resolver manualmente una prueba de escritorio

Les presentamos el siguiente algoritmo en **pseudocódigo**:

```
Algoritmo CalculoDescuento
   Definir precio, descuento, precioFinal Como Real
   Escribir "Ingrese el precio del producto:"
   Leer precio
   Si precio > 100 Entonces
      descuento ← precio * 0.10
   Sino
      descuento ← precio * 0.05
   FinSi
   precioFinal ← precio - descuento
   Escribir "El precio final con descuento es:", precioFinal
FinAlgoritmo
```

Ejercicio:

Realizar una **prueba de escritorio** con los siguientes valores de entrada:

1. `precio = 50`
2. `precio = 120`
3. `precio = 100`

Completar la siguiente tabla con los valores de cada variable durante la ejecución:

Entrada (precio)	Descuento Calculado	Precio Final
50	?	?
120	?	?
100	?	?

PASO 2 ›› Uso de IA para explorar nuevos casos

Ahora, abran **ChatGPT** y pídanle:

🗨️ **"Explica cómo funciona este algoritmo y dame tres ejemplos con valores diferentes a los dados"**

💡 **Reflexión:**

1. ¿Los resultados obtenidos por la IA coinciden con los nuestros?
2. ¿Qué aprendimos al ver otros ejemplos generados por la IA?
3. ¿Cómo podríamos modificar el algoritmo para hacer que sea más flexible o eficiente?

PASO 3 >> Cierre de la actividad

Para finalizar, cada estudiante deberá compartir en el foro de intercambio sus observaciones y reflexiones sobre la prueba de escritorio y los casos generados con la IA.

- Aspectos a comentar:

  - ☐ Comparación entre los resultados obtenidos manualmente y los proporcionados por la IA.
  - ☐ Identificación de posibles errores o diferencias en los cálculos.
  - ☐ Análisis de la utilidad de la IA en la validación de algoritmos.
  - ☐ Propuestas de mejora para hacer el algoritmo más eficiente o adaptable a distintos escenarios.

👉 **Tené en cuenta lo siguiente:** La IA puede ser una herramienta útil para validar y ampliar nuestro aprendizaje, pero es clave que primero intentemos comprender y analizar los algoritmos por nosotros mismos 😀

3. Cierre

En resumen, analizamos cómo los algoritmos permiten resolver problemas de manera estructurada y eficiente. Definimos sus elementos clave: entrada, proceso y salida, y exploramos las fases de análisis, diseño y prueba. También revisamos herramientas como diagramas de flujo y pseudocódigo, así como instrucciones básicas para el control del flujo. Finalmente, destacamos la prueba de escritorio como un paso esencial para validar los algoritmos y subrayamos la importancia de planificar y verificar antes de implementar.

4. Bibliografía

- Joyanes Aguilar, L. (2008). *Fundamentos de programación: Algoritmos, estructura de datos y objetos* (4ª ed.). McGraw-Hill. ISBN 978-84-481-6111-8.
- Bonanata, M. (2003). *Programación y algoritmos: Aprenda a programar en C y Pascal* (Manual Users). M.P. Ediciones. ISBN 987-5261-564.
- Cantone, D. (2003). *La biblia del programador, Implementación y debugging* (Manual Users). M.P. Ediciones. ISBN 987-2299-579.
- OpenAI. (2025). *ChatGPT* (versión GPT-4). Recuperado de https://chat.openai.com