UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

INTRODUCCIÓN A LA PROGRAMACIÓN

UNIDAD 3 – PSEUDOCÓDIGO

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

UNIDAD: 3

PRESENTACIÓN: Pseudocódigo y DF

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

FUNDAMENTOS DE PROGRAMACIÓN

Análisis de problemas

A diario enfrentas problemas y situaciones que debes solucionar de manera óptima bajo diversos contextos, es decir, según las circunstancias que rodean la situación. Un problema se entiende como aquella situación o cuestión, que involucra varios hechos, circunstancias, acciones y obstáculos que impiden alcanzar un objetivo, por lo que es indispensable obtener una solución.

Por otra parte, para solucionar problemas requieres desarrollar ciertas habilidades cognitivas, con el propósito de otorgar una respuesta correcta con base en una situación. En pocas palabras, para solucionar un problema es necesario entenderlo, y para ello se puede empezar por estudiar sus categorías. Los problemas se pueden clasificar en tres tipos, los estructurados, los *semiestructurados* y los no estructurados. A continuación se describe cada uno de estos.

- *Problema estructurado*. Plantea situaciones claras, bien definidas y delimitadas; en estos se puede establecer una solución de forma segura y precisa, la cual puede estar fundamentada en teorías, metodologías y tecnologías.
- *Problema semiestructurado*. Parte de la información disponible es clara y precisa, pero hay incógnitas y dudas en algunos aspectos de la situación.

La labor de quienes tratan de resolver este tipo de problemas es aclarar incógnitas y dudas, con el fin de establecer de forma precisa los requerimientos del programa que se va a generar. Resolver problemas *semiestructurados* presenta dificultades que dependen del grado de las imprecisiones y de la incertidumbre que se tenga, así como de la complejidad de la solución. Los problemas estructurados y *semiestructurados* se pueden resolver utilizando una computadora, por lo que este tipo de problemas serán los que aprenderás a solucionar en este curso.

Problema no estructurado. Su planteamiento es impreciso e incompleto; las relaciones entre sus elementos no son claras, lo que provoca ambigüedad y confusión en la solución requerida. Este tipo de problemas no se pueden resolver utilizando una computadora.

Fases de la solución de problemas

Existen distintas metodologías para la solución de problemas. Todas coinciden en la importancia de plantearse preguntas durante el proceso que sirvan para identificar el problema y buscar alternativas de solución. En este libro de texto se considera que la siguiente metodología es la ideal para solucionar problemas como los que enfrentas a diario. Este procedimiento consta de tres pasos, que son:

1. Análisis profundo del problema
2. Diseño y creación del algoritmo
3. Implementación y verificación del algoritmo

Análisis profundo del problema. Para esta fase necesitas leer un enunciado que describa el problema que se va a solucionar, partiendo de algunas preguntas, y así determinar los teoremas, las fórmulas, las variables y las constantes que se necesitan para dar solución óptima al problema. Las preguntas que pueden plantearse en esta fase son similares a las siguientes: ¿qué necesito resolver?, ¿qué debo

saber para solucionarlo?, ¿cómo podría resolverse?, ¿cuáles son los datos que están involucrados en el problema?, ¿con qué paso se debe empezar?, ¿qué metodología puede aplicarse para resolver el problema?, entre otras.

Al definir los teoremas o fórmulas que se van a usar se determina cuáles y cuántas variables y constantes se utilizarán, y qué procesos van a realizarse para dar solución al problema.

Diseño y creación del algoritmo. Con base en la etapa anterior se definirá la serie de pasos que se van a seguir para dar solución de forma óptima al problema planteado (algoritmo); obviamente, tomando en cuenta todo lo definido en la etapa anterior, como los teoremas, las fórmulas, las variables, las constantes, etcétera.

Implementación y verificación del algoritmo. Finalmente, en esta etapa se usará el algoritmo creado y diseñado en la etapa anterior, y se verificará si el resultado obtenido atiende de manera efectiva la solución del problema; de lo contrario, se debe regresar a la primera etapa para abordarlo de nuevo.

Etapas del diseño y creación de algoritmos

Como lo aprendiste en el tema anterior, el diseño o creación de algoritmos es la segunda fase para la solución de problemas. El algoritmo debe considerar las fórmulas, los teoremas, las variables y demás elementos numéricos para llegar a la solución de uno o varios problemas.

Existen muchas definiciones de algoritmo, que pueden ser diferentes según la óptica de la materia que los aplica; sin embargo, todas las definiciones coinciden en que un algoritmo es un conjunto de pasos, operaciones o acciones finitas, ordenados lógicamente, que permiten solucionar un problema.

Con base en esta definición toda tu vida has utilizado algoritmos para alcanzar objetivos, incluso sin darte cuenta, solo que hasta ahora no habías aplicado la formalidad con la que se deben plantear. Así, todos los algoritmos cuentan con tres etapas principales, que son:

1. Entrada de información. Cualquier algoritmo requiere de la información necesaria para procesarla y solucionar un problema, ya que no se puede partir de la nada, porque la nada no existe, ¿o acaso se podrían sumar dos números que no se conocen?
2. Procesamiento de la información. En esta etapa se aplican fórmulas, teoremas, procesos y demás metodologías que se determinaron en la fase de Análisis profundo del problema, utilizando obviamente la información que fue introducida en la etapa 1.
3. Salida o muestra del resultado. El propósito principal de los algoritmos es solucionar un problema, por lo que se requiere mostrar el resultado, el cual debe atender al problema planteado. Mostrar la solución del problema es el objetivo de esta fase.

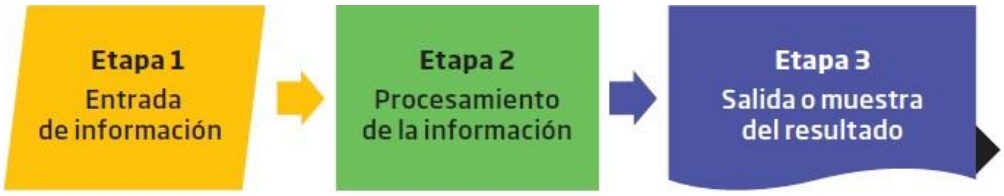


Diagrama de las etapas de un algoritmo.

Además de estas etapas, la creación de un algoritmo también debe cumplir con las siguientes características principales:

1. *Precisión*: la metodología debe estar definida de forma precisa, es decir, no debe existir la ambigüedad en el seguimiento de los pasos del algoritmo creado.
2. *Efectividad*: el algoritmo debe ser efectivo, es decir, las operaciones o procesos deben ser las más adecuadas para solucionar el problema.
3. *Determinismo*: al darle un conjunto de datos idénticos de entrada, el algoritmo debe llegar siempre al mismo resultado.
4. *Finitud*: un algoritmo debe terminar en un número finito de pasos, sin importar su complejidad.

En informática existen varias maneras de categorizar a los algoritmos, pero las más frecuentes consideran cuatro aspectos importantes:

1) ¿Quién? los ejecutará; 2) ¿Qué? signos se usarán; 3) ¿Cuál? es su función; y 4) ¿Cuál? será la estrategia que usará para resolver el problema.

Clasificación de los algoritmos

Los algoritmos se clasifican en informales y computacionales. Los primeros se diseñan para que sean ejecutados por el ser humano y no por una computadora. A diario ejecutas este tipo de algoritmos en diversas actividades, como vestirte, bañarte, preparar comida, ir a la escuela, etcétera. Los algoritmos computacionales se crean tomando en cuenta que una computadora los ejecutará, por lo que los pasos deben ser más precisos que los anteriores, y así aprovechar la velocidad del equipo de cómputo y obtener resultados más confiables.

Como se mencionó en la página anterior, los algoritmos también pueden clasificarse con base en el sistema de signos que utilizan, la función que realizan o la estrategia usada para obtener un resultado. Por lo anterior, se establece que los algoritmos se nombran como sigue:

1. Según su sistema de signos
   a) *Cuantitativos*: cuando se ocupan operaciones numéricas para resolver un problema; por ejemplo, al sumar los precios de dos productos adquiridos en el mercado.
   b) *Cualitativos*: cuando se utilizan instrucciones verbales en la resolución; por ejemplo, al atender las instrucciones de un instructor de manejo cuando muestra la manera de cambiar un neumático.

2. Según su función
   a) *De ordenamiento*: cuando se ordenan datos con base en determinadas normas o reglas.
   b) *De búsqueda*: cuando su tarea principal es encontrar un elemento en una lista determinada de datos.
   c) *De encaminamiento*: cuando se utilizan para buscar datos siguiendo una ruta

determinada y mediante una serie de pasos enlazados, los cuales pueden ser adaptativos o estáticos. Los primeros permiten ajustarse según las circunstancias y los segundos funcionan de forma mecanizada y no permiten adaptación.

3. Según la estrategia usada para llegar a un resultado
   a) *Probabilísticos*: presentan soluciones que pueden ser erróneas o correctas, las cuales se denominan aproximaciones.
   b) *Cotidianos*: se utilizan en la vida diaria y no pueden ser ejecutados en una computadora.
   c) *Heurísticos*: se emplean cuando no se puede llegar a una solución de las maneras tradicionales.
   d) *En escalada*: para llegar a la solución de un problema con estos algoritmos se parte de una solución equivocada, la cual se modifica hasta llegar o estar cerca de la solución correcta.
   e) *Voraz*: el propósito único de este tipo de algoritmos es analizar cada paso como único y darle una solución óptima.
   f) *Deterministas*: estos algoritmos son lineales y predictivos, ya que se conoce exactamente cuáles son los datos o la información que contienen y cómo realizan los procesos para solucionar un problema

Conceptos básicos para la creación de algoritmos

En la creación de los algoritmos también debes utilizar variables y constantes, que están estrechamente relacionadas con las fórmulas o teoremas que se emplearán, pues los algoritmos al ser usados por alguien más pueden darle solución a diversos problemas sin importar los datos que se introduzcan; siempre y cuando el problema sea del mismo tipo del que se basó el diseño del algoritmo.

Variables

El uso de variables no es nuevo para ti, ya que las has usado en asignaturas como Matemáticas, Física o Química. El concepto que conoces sobre variables hasta el momento es que estas son la expresión simbólica de un elemento cuyo valor es desconocido, esto quiere decir que una variable puede adquirir distintos valores. Las variables sin duda alguna son elementos fundamentales para diseñar y crear algoritmos, por tal motivo resulta de vital importancia que aprendas a utilizarlas en este apartado.

En programación, una variable básicamente es un espacio de memoria en el que, de manera temporal, se almacenan datos de varios tipos, los cuales se utilizarán durante alguno de los procesos que se van a ejecutar durante la implementación del algoritmo.
Como podrás darte cuenta, el uso de variables en programación tiene la misma finalidad que en las asignaturas que se mencionaron antes; solo que al programar debes considerar que los recursos necesarios para resolver un problema los obtendrás de una computadora, y que las variables ocuparán un espacio temporal en la memoria RAM.

En programación, el espacio de memoria que ocupará una variable puede ser de longitud fija o variable. Las variables de longitud fija son aquellas cuyo tamaño o espacio no cambia a lo largo de la ejecución de los procesos, pues solo almacenan datos de cualquier tipo, y solo existen algunas excepciones, como las variables de tipo listas, arreglos o cadenas. Por el contrario, las variables de longitud variable modifican su tamaño, pues este tipo de variables se utilizan en las colecciones de datos o arreglos dinámicos.

Las variables usadas en programación deben cumplir ciertas normas o reglas para darles el nombre que las identifica; estas reglas son las siguientes:

1. Evitar el uso de caracteres especiales, acentos, espacios en blanco y símbolos. Si el nombre de la variable se compone de dos palabras se puede usar un guion bajo para unirlas; por ejemplo: numero_cuenta.
2. El nombre de la variable debe escribirse con letras minúsculas; por ejemplo: perímetro. Esto es por factibilidad a la hora de identificar las variables.
3. El nombre de la variable debe tener relación con el dato que almacenará, para ser identificada más rápido y reconocer el tipo de dato que se almacenará en ella. Por ejemplo, si se requiere almacenar dos números en dos variables distintas, las variables podrían llamarse numero1 y numero2.

Existen dos formas de clasificar a las variables informáticas, ya sea por su contenido o por su uso, como se describe a continuación:

Por su contenido:

- *Numéricas*: son aquellas que almacenan números positivos o negativos, incluidos los decimales.
- *Lógicas*: estas solo pueden tener dos significados, es decir, verdadero o falso; se conocen también como booleanas y son usadas para ofrecer el resultado de una comparación.
- *Alfanuméricas*: estas almacenan caracteres alfanuméricos, es decir, letras y números.

Por uso:

- *De trabajo*: son las variables que recibirán la asignación del resultado de un proceso u operación.
- *Contadores*: son variables que llevan el control de las ocasiones en las que se realiza un proceso u operación mientras alguna condición se cumple. Son utilizadas específicamente dentro de los bucles o ciclos.
- *Acumuladores*: son usadas para llevar la suma acumulativa de una serie de valores que se han leído o calculado de manera progresiva.

Constantes

En el proceso de la programación, al igual que en las matemáticas, la constante es un valor que no puede ser modificado o alterado durante el algoritmo. Por tal motivo, a estas les corresponde un tamaño fijo en un área reservada en la memoria principal de la computadora, donde se almacenan valores fijos. Por uso y costumbre, los nombres de las constantes se escriben con mayúsculas. Por ejemplo: PI = 3.1416. Al igual que las variables, las constantes pueden almacenar datos numéricos y alfanuméricos, los cuales no cambiarán.

Tipos de datos fundamentales

Reconocer los tipos de datos te permitirá representar diferente información en el momento de diseñar o crear un algoritmo, sobre todo cuando este sea programable en una computadora.

Existen tres grandes categorías de tipo de datos, las cuales ya has manejado en Excel, estas son numéricos, alfanuméricos y booleanos.

I. *Datos numéricos*. Los datos numéricos, como su nombre lo indica, son aquellos que comprenden números positivos y negativos; estos, a su vez, se dividen en dos categorías: enteros y decimales o de punto flotante.
II. *Datos alfanuméricos*. Se conforman por aquellos caracteres de letras y números que permiten representar información cuyo contenido es descriptivo, es decir, algunos los encontrarás como nombres de personas, direcciones, fechas, horas, etcétera. En ocasiones un dato numérico puede volverse alfanumérico, pero esto implicaría perder su propiedad matemática, es decir, no será posible realizar operaciones con él. El contenido de este tipo de datos, en general, estará encerrado entre comillas.
III. *Booleanos*. Estos datos solo pueden conformarse por dos tipos de valores: uno verdadero y otro falso, los cuales representarán el resultado de comparaciones realizadas durante diferentes procesos.

Como podrás darte cuenta, conocer el tipo de dato que vas a utilizar te ayudará a definir las características de las variables que emplearás en el algoritmo, pues se encuentran estrechamente relacionados.

Estructuras de control

Las estructuras de control en programación son aquellas que permiten controlar el orden de ejecución de las instrucciones del algoritmo empleado para la solución de un problema en específico. Las estructuras de control usan comparaciones como parámetro principal, y para poder especificar estas se usan operadores lógicos y los llamados de condición. El siguiente cuadro muestra los operadores lógicos y de condición más utilizados.

Operadores de comparación	
Operador	**Significado**
<	Menor que
>	Mayor que
< =	Menor o igual
> =	Mayor o igual
< > o !=	Diferente
= =	Igual
Operadores lógicos	
Operador	**Significado**
AND o && o Y	Y (unión)
OR o || u O	O (disyunción)
NOT o ! o NO	Negación

Estructuras de control secuencial

Este tipo de estructuras de control se ejecuta conforme aparece en el algoritmo, es decir, en orden secuencial, del paso 1 al 2, al 3 y así sucesivamente.

Para comprender mejor este tipo de estructuras, piensa en la manera en que diseñaste y creaste un algoritmo para solucionar los problemas planteados en el tema anterior. Además, realiza las actividades que se describen a continuación con el apoyo de tu profesor. Cuando se quiere mostrar los valores incluidos en las variables es necesario concatenarlas usando el signo +.

Ejemplo 1.1.1

Problema: obtener el perímetro y el área de un cuadrado del cual no se conocen sus dimensiones.
Solución: diseñar un algoritmo con base en las fases de solución de problemas, usando estructuras de control secuencial.

Fase 1: se requiere conocer las fórmulas para calcular el perímetro y el área de un cuadrado. Con base en la definición de un cuadrado, la medida de sus cuatro lados es la misma. Por tanto:

- Fórmula del perímetro: P = 4*medida del lado.
- Fórmula del área: A = (medida del lado)^2.
- Variables: *perímetro, area* y *m_lado*.

Estructuras de control selectivo

Este tipo de estructuras se usa cuando la continuación del desarrollo del algoritmo debe tomar una decisión para continuar con un proceso o una ruta alternativa. Estas decisiones se basan en la valoración de una o más condiciones, las cuales señalan hacia dónde se modificará el sentido del algoritmo. Las estructuras de control selectivo se dividen en tres grandes tipos: simples, dobles y múltiples.

Las estructuras de control selectivo simple tienen como característica principal que se componen por una sola condición, es decir, el que se ejecute uno o más procesos va a depender si la condición evaluada se cumple; de lo contrario no se realiza ningún proceso o acción. Este tipo de estructuras expresan el algoritmo de la siguiente manera:

Si (condición(es)) entonces
..... Proceso(s)
Fin del Si

Las estructuras de control selectivo doble, al igual que las simples, están compuestas por una sola condición, pero en estas habrá dos caminos que tomar con relación a los procesos del algoritmo; uno será si la condición evaluada se cumple y el otro si no se cumple. En otras palabras, si se cumple o no se realizarán distintos procesos. Este tipo de estructuras se expresan así:

Si (condición(es)) entonces
..... Proceso(s) De lo contrario
..... Proceso(S) Fin del Si

Ejemplo 1.1.2

Problema: conocer si un número es positivo o negativo.
Solución: diseñar dos algoritmos, uno para solucionar el problema usando estructuras de control selectivo simple y otro mediante estructuras de control selectivo doble.

Por último, las estructuras de control selectivo múltiples, también conocidas como anidadas, se dividen para su estudio en dos tipos, las estructuras de control selectivo simple anidado y las selectivas de caso.
Las primeras están formadas por estructuras de control selectivo simple anidadas con la instrucción "de lo contrario Si"; estas estructuras se pueden identificar en los algoritmos de la siguiente manera:

Si (condición(es)) entonces
..... Proceso(s)
De lo contrario Si (condición(es)) entonces
..... Proceso(s) De lo contrario
..... Proceso(s) Fin del Si

Las estructuras de control selectivo de caso se identifican en los algoritmos con la expresión En caso de, las cuales toman como parámetro una variable tipo opción, la cual también sirve de guía para el flujo del algoritmo con relación a los procesos por realizar, y siempre tendrá una opción por defecto. En un algoritmo se identifican de la siguiente manera:

En caso de (opción) Caso valor1:
.... Proceso(s) Interrumpir Caso valor2:
..... Proceso(s) Interrumpir Caso defecto:
..... Proceso(s) Fin del caso

Ejemplo 1.1.3

Problema: conocer si un número es positivo, negativo o igual a cero.
Solución: diseñar un algoritmo para solucionar el problema utilizando estructuras de control selectivo simple anidado.

Ejemplo 1.1.4

Problema: conocer el nombre del día de la semana con solo introducir el número de día que es.
Solución: diseñar un algoritmo para solucionar el problema utilizando estructuras de control selectivo de caso.

Estructuras de control repetitivo

Las estructuras de control repetitivo, también llamadas de iteración, permiten ejecutar uno o varios procesos de manera repetida. En algunas referencias bibliográficas las podrás encontrar con el nombre de ciclos o bucles porque estas estructuras de control permiten que se repita una serie de procesos. Para diseñar y crear este tipo de estructuras es necesario considerar dos aspectos muy importantes: ¿cuántas veces se repite el ciclo?, y ¿cuál será el contenido o cuerpo del ciclo? En este tipo de estructuras de control siempre se usan las variables del tipo contador, las cuales juegan un papel muy importante en el funcionamiento de los ciclos o bucles porque las condicionantes con las que se trabajan los algoritmos responderán al contenido de los contadores; es decir, sin un contador

los ciclos quedarán ciclados, valga la redundancia, pues nunca se podría salir de ellos; por tanto, se debe tener cuidado de siempre incluir contadores en los ciclos.

El contador puede comportarse de dos maneras, ya sea en aumento o en decremento. El comportamiento de los contadores depende de los procesos que necesites ejecutar dentro de los ciclos. Los contadores se identifican con variables cuyo nombre empiezan con las letras i, j, k, l, m, n. Además de los contadores, en ocasiones usarás variables tipo bandera, conocidas también como booleanas, cuyo propósito es servir de interruptor dentro del ciclo. Estas variables deben estar inicializadas en 0 antes de entrar al ciclo; una vez dentro de este, según el proceso o condición, puede cambiar su valor a 1, ya que 0 es false (falso o apagado) y 1 es true (verdadero o encendido).

Existen tres principales estructuras de control repetitivas, que son: mientras, para y haz mientras; de las cuales solo se aplicarán las dos primeras.

La estructura Mientras se conoce también como *while* en el ámbito de la programación, y se utiliza principalmente cuando no se conoce el número de repeticiones o vueltas que debe dar el bucle al realizar los procesos establecidos.
Este número depende de las proposiciones o condiciones dentro del ciclo, es decir, los procesos que se encuentren en el cuerpo del ciclo se ejecutarán siempre y cuando su condición siga siendo verdadera; en el momento en el que esta no se cumpla el ciclo será interrumpido y el flujo del algoritmo continuará con la siguiente secuencia de pasos.

La estructura de este ciclo consta de dos partes, que son: el ciclo (conjunto de procesos o instrucciones que se ejecutarán de manera iterativa), y la condición (elemento que se evaluará para permanecer o salir de la ejecución del ciclo en sí). En los algoritmos, la estructura Mientras se identificará como se muestra a continuación:

```
Mientras (condición(es))
..... Proceso(s)
Fin del Mientras
```

La estructura de control repetitivo Haz Mientras se conoce también como do *while*. Su funcionamiento es muy similar al del ciclo mientras, solo que en esta los procesos dentro del cuerpo del ciclo se ejecutarán al menos una vez antes de que la condición sea evaluada, es decir, la estructura de este bucle permite una ejecución al menos antes de salirse por completo si la condición no se llega a cumplir.

Al igual que en la estructura Mientras (*while*), Haz Mientras consta de dos partes, que son el ciclo (conjunto de procesos o instrucciones que se ejecutarán de manera iterativa), y la condición (elemento que se evaluará para permanecer o salir de la ejecución del ciclo en sí). En los algoritmos, la estructura Haz Mientras se identifica con la estructura siguiente:

```
Haz
.... Proceso(s)
Mientras (condición(es))
```

La estructura Para se llama también *for*. A diferencia de las estructuras mencionadas antes, esta se usa cuando ya se encuentra definido el número de veces que se ejecutará el ciclo, aunque en

ocasiones se puede utilizar para hacer ciclos infinitos.
Para es la estructura de control repetitivo más usada entre los desarrolladores debido a su sencilla estructura. Su expresión está compuesta por cuatro partes: la tarea de inicialización (indica a partir de qué número se empezará a contar), la condición (elemento que se evaluará para permanecer o salir de la ejecución del ciclo en sí), la tarea final por vuelta (define el comportamiento de la variable contador dentro del ciclo) y, por último, el ciclo (conjunto de procesos o instrucciones que se ejecutarán de manera iterativa). En los algoritmos se encontrará expresada de la siguiente manera:

Para (tarea_de_inicialización; condición; tarea_final_por_vuelta)
.... Proceso(s)
Fin del Para

Ejemplo 1.1.5

Problema: visualizar la tabla de multiplicar de cualquier número entero positivo con resultados que abarquen hasta ser multiplicado por 15.
Solución: diseñar un algoritmo para solucionar el problema utilizando la estructura de control repetitivo *Mientras*.

Ejemplo 1.1.6

Problema: visualizar la tabla de multiplicar de cualquier número entero positivo con resultados que abarquen hasta ser multiplicado por 15.
Solución: diseñar un algoritmo para solucionar el problema utilizando la estructura de control repetitivo *Para*.

Diagramas de flujo

¿Qué es un diagrama de flujo?
Un diagrama de flujo es un organizador gráfico de tipo secuencial, cuyo propósito principal es la representación gráfica de procesos e ideas, específicamente de algoritmos como los que diseñaste y creaste en la secuencia didáctica anterior. Los diagramas de flujo facilitan la lectura de los algoritmos porque en ellos se especifica la ruta directa que deben tomar los procesos de la solución del problema.

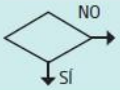
Los diagramas de flujo no son utilizados solo por las disciplinas de la informática, como la programación, sino que también se usan para esquematizar cualquier tema de análisis; son usados en la educación, la economía y la industria, por ejemplo. Por tal motivo, este tipo de organizadores gráficos secuenciales son muy diversos, aunque tienen en común un aspecto muy importante, todos deben mostrar con claridad el vínculo entre los conceptos enmarcados y las ideas principales y secundarias.

Para que cada diagrama de flujo cobre el sentido adecuado debe tener una trayectoria bien definida hacia la solución de un problema, partiendo de un inicio y llegando a un final, tal y como sucede con los algoritmos, pues estos serán la fuente de los diagramas de flujo que crearás en esta secuencia didáctica.

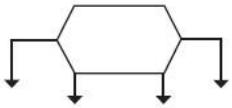
En este libro se usarán los diagramas de flujo para expresar de forma gráfica la secuencia o el flujo de un algoritmo; para lograr su propósito, este tipo de esquemas o diagramas se valen de diferentes símbolos, por lo que será el siguiente tema por estudiar.

Simbología de los diagramas de flujo

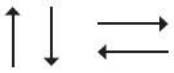
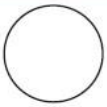
El siguiente cuadro muestra los símbolos más utilizados en la diagramación de los algoritmos.

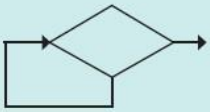
Símbolo	Significado	Instrucción algorítmica
	Inicio y fin del diagrama.	`Inicio` `Fin`
	Entrada de datos, es decir, expresa el sentido de la lectura.	`Asignar respuesta a …`
	Representa un proceso o diversas operaciones por realizar.	`Asignar proceso a …` `Asignar resultado a …`
	Representa una toma de decisión. En el interior del símbolo debe expresarse una condición que, según su resultado, es el camino que tomará el flujo del diagrama.	`Si (condición(es))` `Fin del Si`

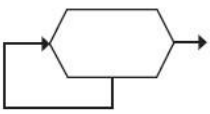
	Representa una toma de decisión doble, es decir, Si entonces - De lo contrario. En su interior debe estar la condición y si el resultado es verdadero el flujo va hacia la izquierda y si es falso a la derecha.	`Si (condición(es))`
		`De lo contrario`
		`Fin del Si`
		
		`Si (condición(es))`
		`De lo contrario Si`
		`(condición(es))`
		`De lo contrario`
		`Fin del Si`

	Simboliza toma de decisiones múltiples o selección de casos. Dentro del símbolo debe estar el selector y, dependiendo de su valor, será el camino que tomará el flujo del diagrama. **Instrucción algorítmica:** En caso de	`En caso de (opción)` `Fin del caso de`

	Representación simbólica de la salida de resultados, expresa la escritura de datos obtenidos.
	Instrucciones algorítmicas:
	Decir, Expresar y Preguntar.
		`Decir`
		`Expresar`
		`Preguntar`

	Son usados para definir la dirección del flujo del diagrama.	
	Conector utilizado dentro de una misma página.	
	Conector usado entre páginas diferentes.	

	Representación de los ciclos o bucles *Mientras* (While) y *Haz Mientras* (Do while). Dentro del símbolo debe ir la condición que hará que el ciclo se ejecute mientras esta se cumpla. Dentro de los procesos del bucle debe estar el incremento o decremento del contador de eventos para lograr romper el ciclo.	`Mientras (condición(es))`
		`Fin del mientras`
		
		`Haz`
		
		`Mientras (condición(es))`

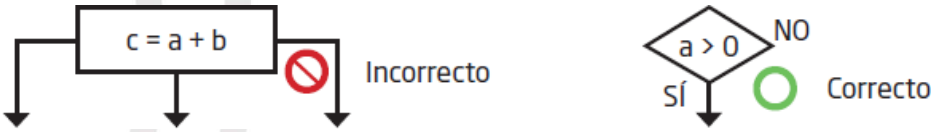
	Representación del ciclo o bucle *Para* (For). El símbolo está dividido en tres partes, la primera determina a partir de cuándo o de qué valor inicia el ciclo; la segunda incluye la condición que hace que el ciclo se ejecute; y la tercera parte determina la menera en que el contador de eventos del ciclo se irá incrementando hasta llegar a la condición planteada y se rompa el bucle.	`Para (tarea_de_`
		`inicialización;`
		`condición;`
		`tarea_final_por_vuelta)`
		
		`Fin del Para`

Reglas para la construcción de diagramas de flujo

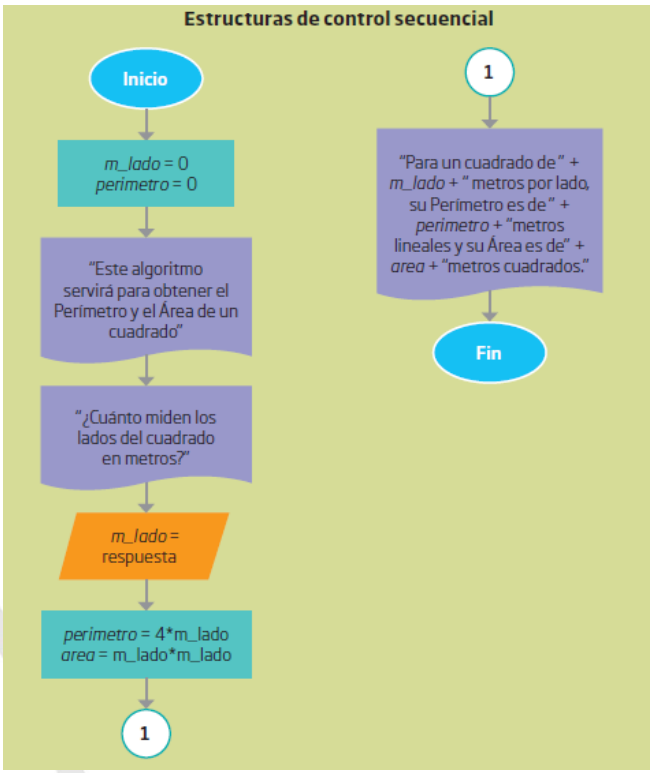
Para realizar un diagrama de flujo de manera adecuada debes tomar en cuenta las siguientes reglas o normas:

- Todo diagrama de flujo debe tener un inicio y un fin, no pueden quedar instrucciones o decisiones abiertas.
- Su elaboración debe ajustarse a las etapas de la creación y de diseño de un algoritmo.
- Se escriben en forma vertical, de arriba hacia abajo y de izquierda a derecha.
- Todos y cada uno de los símbolos deberán contener dentro de ellos la función o proceso que deberá realizarse en ese paso, como se muestra a continuación:

- Ningún elemento podrá tener más de una salida, con excepción de los relacionados con las decisiones.



- Las líneas del diagrama deben cruzarse. Observa la manera correcta y la incorrecta de elaborarlo. Ejemplo:



¿Qué es un pseudocódigo?

El proceso de desarrollo de programas informáticos contempla una fase que resulta muy importante, que es la creación del pseudocódigo, nombre con el que se conoce a la forma de escribir los pasos que va a ejecutar un programa, la cual debe cumplir una estructura similar a la del lenguaje de programación.

El propósito principal de diseñar y crear un pseudocódigo es darle la formalidad necesaria a los algoritmos y diagramas de flujo, para después codificarlos en el lenguaje de programación que se requieren para su ejecución. Para lograrlo, se deben seguir ciertas reglas, las cuales conocerás a continuación.
Reglas para la creación de un pseudocódigo Comúnmente, los pseudocódigos se escriben a mano o en algún procesador de texto, ya que realmente terminan siendo casi igual a los algoritmos, solo que se les agregan algunos componentes que determinan su función, como pueden ser los siguientes:

a) Después de la palabra Inicio se debe colocar el carácter '{'.
b) Antes de la palabra Fin se escribe el símbolo '}'.
c) Se debe agregar el símbolo ';' al terminar cada sentencia o proceso a realizar.
d) Se coloca el símbolo '{' al iniciar las estructuras de control y el símbolo } al finalizar, esta última sustituye la leyenda usada en el algoritmo que define el fin de la estructura.

Con base en estas diferencias, analiza el siguiente comparativo de escritura entre un algoritmo y un pseudocódigo:

Comparación de escrituras entre...	
algoritmo	**pseudocódigo**
Si (condición(es)) entonces	Si (condición(es)){
... Proceso(s)	... Proceso(s);
Fin del *Si*	}

Ejemplo: Realizar el pseudocódigo de un programa que permita calcular el área de un rectángulo. Se debe introducir la base y la altura para poder realizar el cálculo…

Programa; área
Entorno: BASE, ALTURA, AREA son número enteros
Algoritmo:
    escribir “Introduzca la base y la altura” leer BASE, ALTURA
    calcular AREA = BASE * ALTURA
    escribir “El área del rectángulo es “AREA
Finprograma