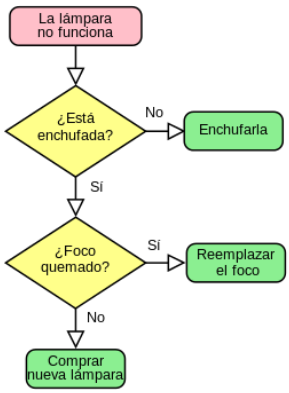
2º de Bachillerato

Tecnologías de la Información y Comunicación Contenidos

Conceptos básicos de programación: Planteando un programa: Diagramas de flujo y pseudocódigo



Algoritmo diseñado con Diagrama de flujo

Imagen en wikipedia de Booyabazooka, Licencia CC

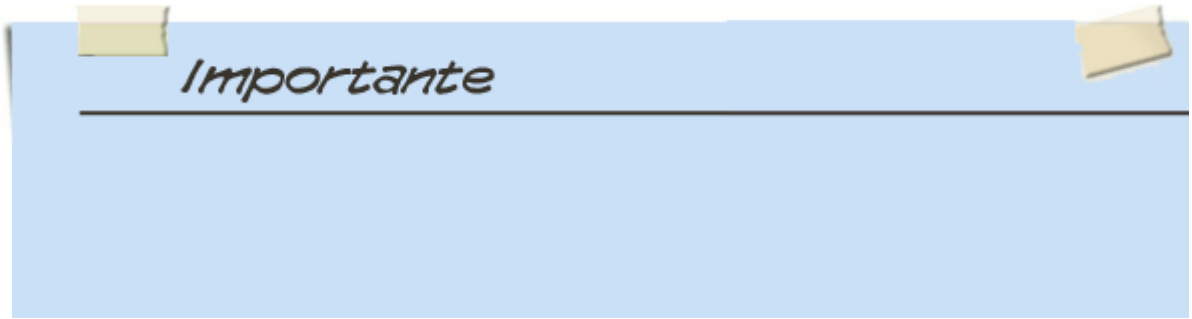
Para resolver problemas en Informática, como en otras disciplinas de nuestra vida, se necesita una **metodología** que permita llegar a las soluciones adecuadas. Esta metodología se fundamenta en los algoritmos.

Un algoritmo es un conjunto finito de pasos que, realizados en un determinado orden, permite resolver un problema determinado. Un ejemplo típico es una receta de cocina, con la que, después de realizar todos sus pasos o fases, en un determinado orden, conseguiremos un suculento bocado. Cada uno de los paso de un algoritmo se denomina **sentencia o instrucción**. Los algoritmos se pueden expresar de varias formas, entre las que destacan dos: **los diagramas de flujo y el pseudocódigo**. Esta última herramienta es una de las más usadas.

A las soluciones creadas en un ordenador se les conoce como **programas** y no son más que una serie de operaciones que realiza la computadora para llegar a un resultado, partiendo de un grupo de datos específicos.

Para poder realizar programas, además de conocer la metodología mencionada, también debemos conocer, de manera específica, las funciones que el ordenador puede realizar y las formas en que se pueden manejar los elementos que hay en el mismo.

Un lenguaje de programación es un conjunto de caracteres, símbolos y reglas que, escritos de una determinada manera (sintaxis), permite a las personas comunicarse con el ordenador. Los lenguajes de programación tienen un conjunto de instrucciones que nos permiten realizar operaciones de diverso tipo: de entrada y salida de datos, de cálculo, de manipulación de textos, lógicas, de comparación y de almacenamiento o recuperación de datos.

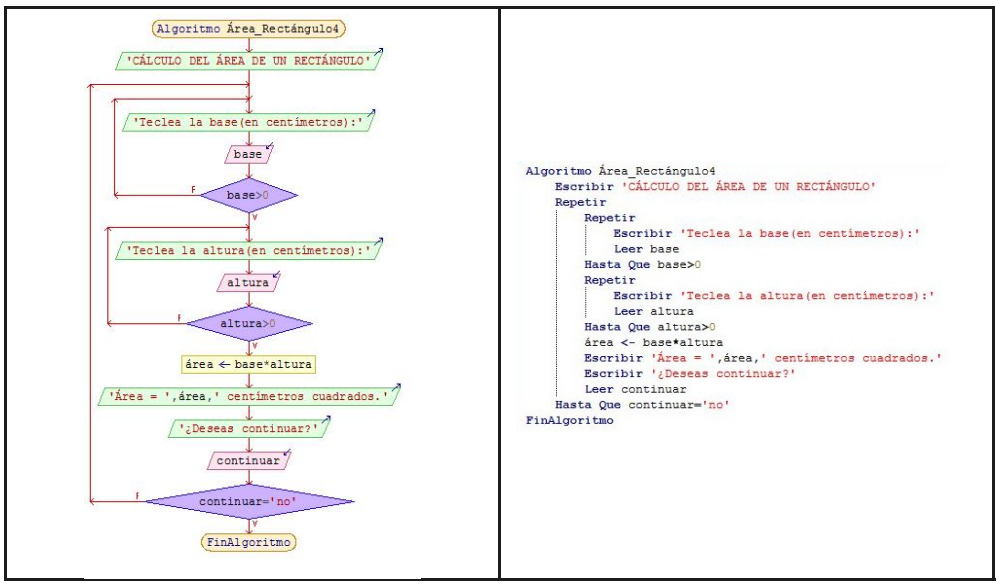


Cuando se va a construir un programa informático, lo habitual es diseñar primero el algoritmo en forma de **diagrama de flujo** y/o **pseudocódigo**, para luego traducirlo al lenguaje de programación concreto a utilizar. Se hace de esta forma porque así, el algoritmo es independiente del lenguaje a usar, pudiéndose traducir en cualquier momento a otros lenguajes de programación si fuese necesario.

1. ¿Qué vas a aprender en este tema?

Al finalizar el estudio de este tema, conseguirás no sólo entender algoritmos expresados en forma de diagramas de flujo o en pseudocódigo, sino también poder elaborar pequeños programas que el propio ordenador será capaz de entender y ejecutar.

Un ejemplo de ello lo puedes ver a continuación:



Todo ello podrás ponerlo en práctica gracias a herramientas software de uso libre y gratuitas. Una de estas herramientas, muy recomendable, es **PSeInt**.

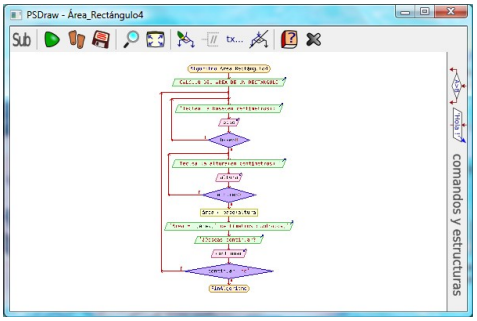


Imagen de creación propia bajo licencia Creative Commons

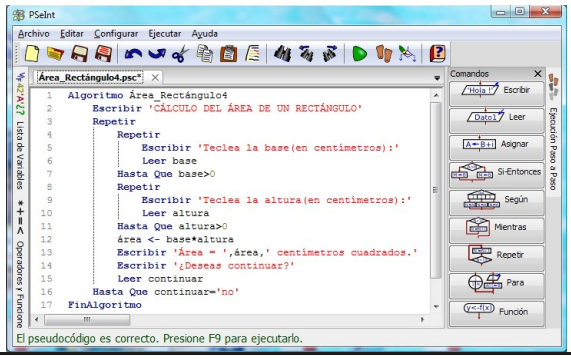


Imagen de creación propia bajo licencia Creative Commons

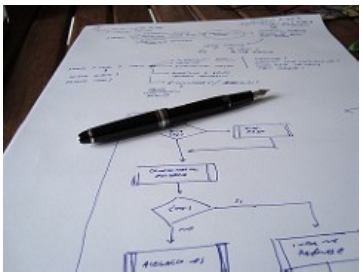


Imagen en Flick de Ángel Medinilla bajo licenciaCC

Un Diagrama de Flujo no es más que la representación gráfica de un algoritmo. Los diagramas de flujo son esquemas que describen la secuencia de pasos o fases de un proceso y emplean símbolos gráficos para representarlos. Estos símbolos van unidos o conectados por medio de flechas para indicar la secuencia de las operaciones, de ahí que sean llamados diagramas de flujo. La simbología utilizada para la elaboración de diagramas de flujo es variable y debe ajustarse a unas normas concretas, lo más universales posible para conseguir que sean entendibles por cualquier persona.

Su correcta construcción es muy importante porque a partir del mismo se implementará un programa informático usando un lenguaje de programación determinado, por tanto, si el diagrama de flujo está mal diseñado, el programa fallará también. Si el Diagrama de Flujo está completo y es correcto, el paso del mismo a un Lenguaje de Programación es relativamente simple y directo.

En un diagrama de flujo podremos distinguir, además de los datos necesarios para el proceso, los siguientes aspectos:

- Los caminos y direcciones que los datos deben seguir.
- El origen de los datos.
- El destino de los mismos.
- Las transformaciones que sufren.
- Etc.

Para saber más

Tipos de Diagramas de Flujo

Existen muchos tipos y clasificaciones distintas de Diagramas de Flujo, cada autor considera matices diferentes que dan lugar a esta diversidad pero los criterios de clasificación suelen ser comunes en todas ellas, destacando sobre todo las clasificaciones según el formato y según el objetivo perseguido.

Así, **según el formato**, podemos distinguir entre:

- Diagramas de flujo verticales y horizontales, que son aquellos en que el flujo de la información circula en esos sentidos respectivamente, es decir de arriba a abajo o de izquiera a derecha.
- Panorámicos o tabulares, que son aquellos que se presentan en un sólo folio y contiene columnas representando distintos aspectos (actividades, puestos de trabajo, usuarios y personas involucradas,...).
- Arquitectónicos, que ponen el énfasis en el espacio por donde circulan los datos.

Según el objetivo perseguido destacan:

- Diagramas de flujo de forma, en los que no se detalla ninguna descripción o la mínima posible.

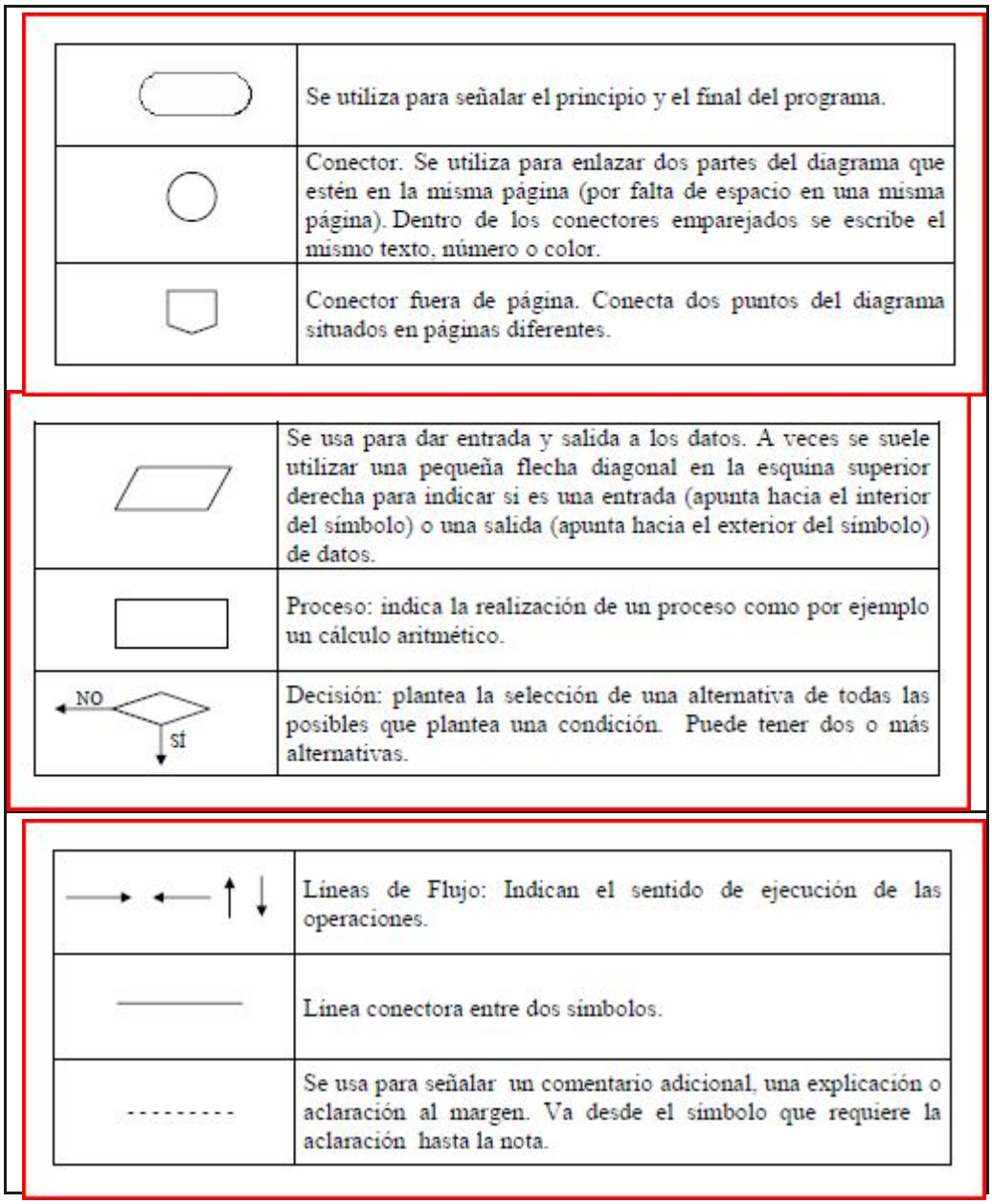
refleja también para qué sirve el procedimiento.

- De espacio, que indican los lugares por donde circulan los datos.
- De métodos, en los que se indica la persona que realiza cada función y fundamentalmente cómo se desarrolla cada una.
- De ilustraciones y textos, que pretenden manejar toda la información con ayuda de dibujos y textos explicativos que faciliten y completen el algoritmo.
- Asistidos por ordenador, que son los controlados por software.
- Combinados, en los que se da una mezcla de varios tipos de diagramas.

2.1 Elementos de un diagrama de flujo

Si te dedicas a hacer una búsqueda de los distintos símbolos utilizados en los diagramas de flujo, verás que existe una gran variedad. Algunos de ellos ya están en desuso (lógico si observamos el ritmo vertiginoso de la innovación tecnológica en los últimos años), otros tienen significados parecidos o idénticos y se utilizan por tanto para representar las mismas acciones, pero hay unos pocos que, como podrás apreciar, no solo siguen vigentes, sino que además, cubren bastante bien un grado de representación bastante aceptable.

No es cometido de este tema hacer una recopilación extensa de esta simbología, sino más bien todo lo contrario, es por ello por lo que nos centraremos en ese conjunto de símbolos reducido pero a la vez de uso amplio.



	Se utiliza para señalar el principio y el final del programa.
	Conector. Se utiliza para enlazar dos partes del diagrama que estén en la misma página (por falta de espacio en una misma página). Dentro de los conectores emparejados se escribe el mismo texto, número o color.
	Conector fuera de página. Conecta dos puntos del diagrama situados en páginas diferentes.

	Se usa para dar entrada y salida a los datos. A veces se suele utilizar una pequeña flecha diagonal en la esquina superior derecha para indicar si es una entrada (apunta hacia el interior del símbolo) o una salida (apunta hacia el exterior del símbolo) de datos.
	Proceso: indica la realización de un proceso como por ejemplo un cálculo aritmético.
NO / SI	Decisión: plantea la selección de una alternativa de todas las posibles que plantea una condición. Puede tener dos o más alternativas.

	Líneas de Flujo: Indican el sentido de ejecución de las operaciones.
	Línea conectora entre dos símbolos.
	Se usa para señalar un comentario adicional, una explicación o aclaración al margen. Va desde el símbolo que requiere la aclaración hasta la nota.

	Salida de información por pantalla.
	Salida de información por impresora.
	Disco o dispositivo de almacenamiento.

Aunque este conjunto de símbolos es bastante estándar, puede que te encuentres otros símbolos para representar las mismas acciones (por ejemplo que el de inicio de un diagrama sea un rectángulo con doble línea en sus extremos), o incluso los mismos símbolos pero con ligeros matices (por ejemplo en los rombos que simbolizan los puntos de decisión, en lugar de llevar en sus salidas un "SI" y un "NO" podría aparecer "VERDADERO" y "FALSO", o incluso sus formas abreviadas "V" y "F").

Lo importante en cualquier caso es que se reconozca la acción básica que representa cada símbolo.

2.2 Construcción de un diagrama de flujo

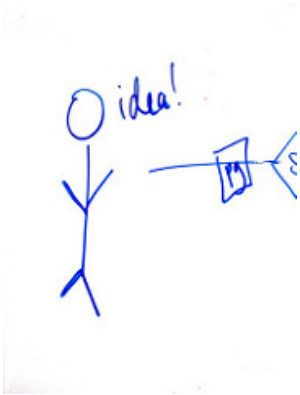


Imagen en Flickr de Quinn Dombrowski con CC

Para estructurar de manera adecuada un diagrama de flujo, es conveniente que, con anterioridad, tengamos en cuenta una serie de aspectos, entre los que destacan siguientes:

- Pensar las ideas esenciales que el diagrama de flujo debe incluir.
- Definir el objetivo principal del diagrama de flujo.
- Fijar las limitaciones del algoritmo.

Una vez establecido el contexto, construiremos el diagrama de flujo haciendo hincapié en las reglas siguientes:

1. Proporcionar un nombre adecuado al diagrama, que sea representativo de la tarea que el algoritmo realiza.
2. Debe tener obligatoriamente un comienzo y un final.
3. Plasmar los pasos del algoritmo, por orden, de arriba a abajo y de izquierda a derecha, representando cada uno con el símbolo adecuado e incluyendo en el interior del mismo una breve descripción del proceso a realizar en ese paso. Se debe tener en cuenta que cada símbolo, excepto las líneas de flujo, debe llevar en su interior información, no puede haber ninguno vacío.
4. Cada símbolo, a excepción de los de decisión, sólo puede tener una línea de flujo de salida.
5. Las líneas de flujo no deben cruzarse.
6. Un elemento del diagrama no puede tener más de una salida si no es un elemento de decisión
7. Es muy importante detectar los puntos de decisión, que son aquellos en los que la ejecución del programa puede tomar una dirección u otra.
8. Si es posible, el diagrama de flujo se mostrará en una sóla página. Cuando no sea posible deben usarse conectores.

Para saber más

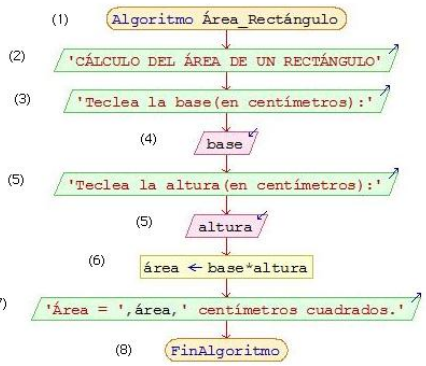
Si el algoritmo a representar mediante un diagrama de flujo es extenso, lo habitual es dividir el diagrama en varios, de tal manera que el final del primero coincida con el comienzo del segundo, el final del segundo con el comienzo del tercero, y así sucesivamente hasta completar la representación del algoritmo al completo. Es decir, que la idea es dividir el diagrama completo en subdiagramas que luego irán concatenados entre sí.

2.2.1 Un pequeño ejemplo

Imaginemos que necesitamos un programa que calcule el área de un rectángulo a partir de la base y la altura del mismo, datos que serán solicitados por el programa al usuario.

El algoritmo se podría enfocar de la siguiente manera:

1. Hay que ponerle nombre al diagrama de flujo, en consonancia con lo que el algoritmo conseguirá realizar. "Área_Rectángulo" podría ser un buen nombre, que se incluirá en el símbolo de comienzo del diagrama. Esta acción no tendría ningún efecto visual en pantalla, es decir que sería transparente para el usuario.
2. Seguidamente se muestra en pantalla una frase informativa indicando que se calculará el área de un rectángulo.
3. El paso siguiente consistirá en informar al usuario que debe teclear la base del rectángulo en centímetros.
4. En este momento, el ordenador debe esperar a que el usuario teclee el dato correspondiente a la base del rectángulo. Aquí, una vez que el usuario introduzca la base, el valor tecleado se almacenará en una zona de la memoria que el programa llamará "base", de tal manera que, cada vez que se quiera utilizar a lo largo de nuestro programa dicho valor, bastará con especificar su nombre (base). Eso hará que el mismo programa nos sirva para cualquier rectángulo al que le queramos calcular el área, independientemente de la base que tenga.
5. El algoritmo actuará de igual forma para la altura del rectángulo (primero es conveniente que muestre en pantalla una frase pidiéndole al usuario que teclee la altura, y posteriormente deberá esperar a que la altura sea introducida por teclado). En este caso, el valor se almacenará en memoria pero esta vez se le asignará el nombre "altura".
6. Una vez que el programa conozca ambos datos, la base y la altura, estará en disposición de calcular el área del rectángulo. Simplemente deberá aplicar la fórmula correspondiente (el área de un rectángulo es igual a la base por la altura). En este paso, el algoritmo realizará el cálculo multiplicando el valor que se almacenó en memoria con el nombre base y el valor que se almacenó en memoria con el nombre altura. El resultado será almacenado en otra zona de memoria a la que se le llamará "área".
7. A continuación se mostrará en pantalla el valor del área calculada. Para ello, se formará una frase que tiene 3 partes: la primera y la última son textos fijos ("Área = " y "centímetros cuadrados", nótese que van entrecomillados), y la parte central que corresponde al valor del área (referenciado con el nombre área, que es como se llamó a la zona de memoria donde se almacenó dicho cálculo).
8. Finalmente se indicará el fin del algoritmo.



En este ejemplo hay varios **aspectos que mencionar:**

- En primer lugar, hay que decir que los números que aparecen a la izquierda de cada símbolo del diagrama de flujo, son meramente didácticos, solo están para hacer más entendible la explicación y que se pueda apreciar mejor a qué paso corresponde cada uno de los elementos del diagrama, por tanto, no se suelen poner.

- En segundo lugar, observar que los símbolos correspondientes a romboides muestran tanto las entradas (por teclado) como las salidas (por pantalla) de información, pero se aprecia el uso de las pequeñas flechas en la esquina superior derecha para indicar si son entradas (apuntan hacia el interior del romboide) o salidas (apuntan hacia el exterior del romboide).
- Otra apreciación sería a nivel de lo que se muestra por pantalla. Cuando se quiere que aparezca un texto fijo en pantalla (siempre el mismo), bastaría con entrecomillarlo ('Teclea la base en centímetros:'), sin embargo, cuando se desea mostrar por pantalla el valor almacenado en una zona de memoria, se especifica el nombre de esa zona de memoria pero sin entrecomillarlo (es lo que ocurre en la frase que muestra el resultado calculado, en la que se aprecia que la palabra área del centro no va entre comillas, ya que se refiere al valor almacenado en memoria).

Para saber más

A estos valores que pueden ser distintos cada vez que se ejecuta el programa y que son almacenados en memoria y referenciados con un nombre, se les llama **"variables"**.

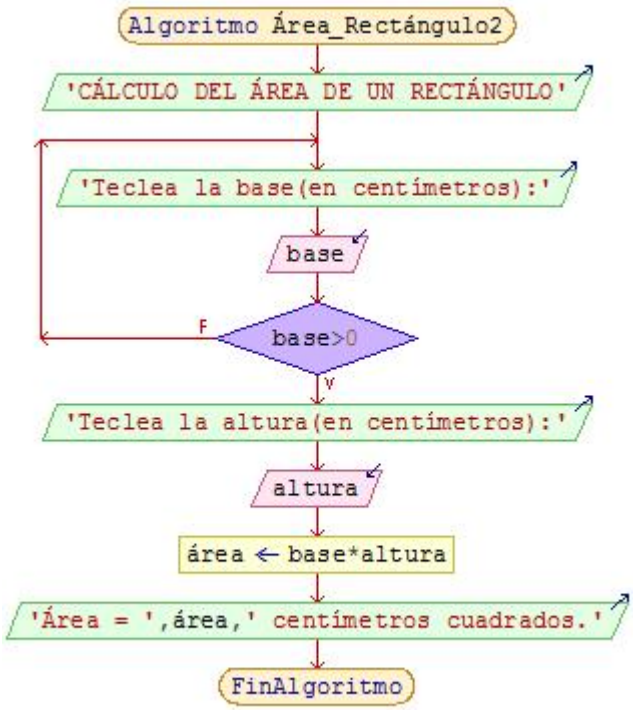
En contenidos posteriores conocerás algunos detalles más sobre estos elementos fundamentales en programación y sobre otros parecidos.

En el algoritmo anterior, el usuario tendrá el poder de introducir tanto la base como la altura que desee cada vez que ejecute el programa. ¿Qué pasaría si teclea una base negativa?, en teoría, un rectángulo con base negativa no existe, ¿crees que el programa se pararía al introducir la base negativa o continuaría hasta el final? Y si continuase hasta el final, ¿qué resultado mostraría como área?. En nuestro caso, la respuesta es simple, el programa no se detendría y al final mostraría un valor para el área que sería negativo, lo cual es incongruente pues no existen los rectángulos con áreas negativas. Igual ocurriría con una base igual a cero.

¿Se te ocurre cómo solucionar el problema?... Efectivamente, la respuesta la tienes en los puntos de decisión (el símbolo del rombo).

Si de modificara ligeramente el diagrama utilizando para ello un punto de decisión (un rombo) justo después de que el usuario teclease el valor para la base, se podría cambiar el flujo del programa en dos sentidos dependiendo del valor de la base.

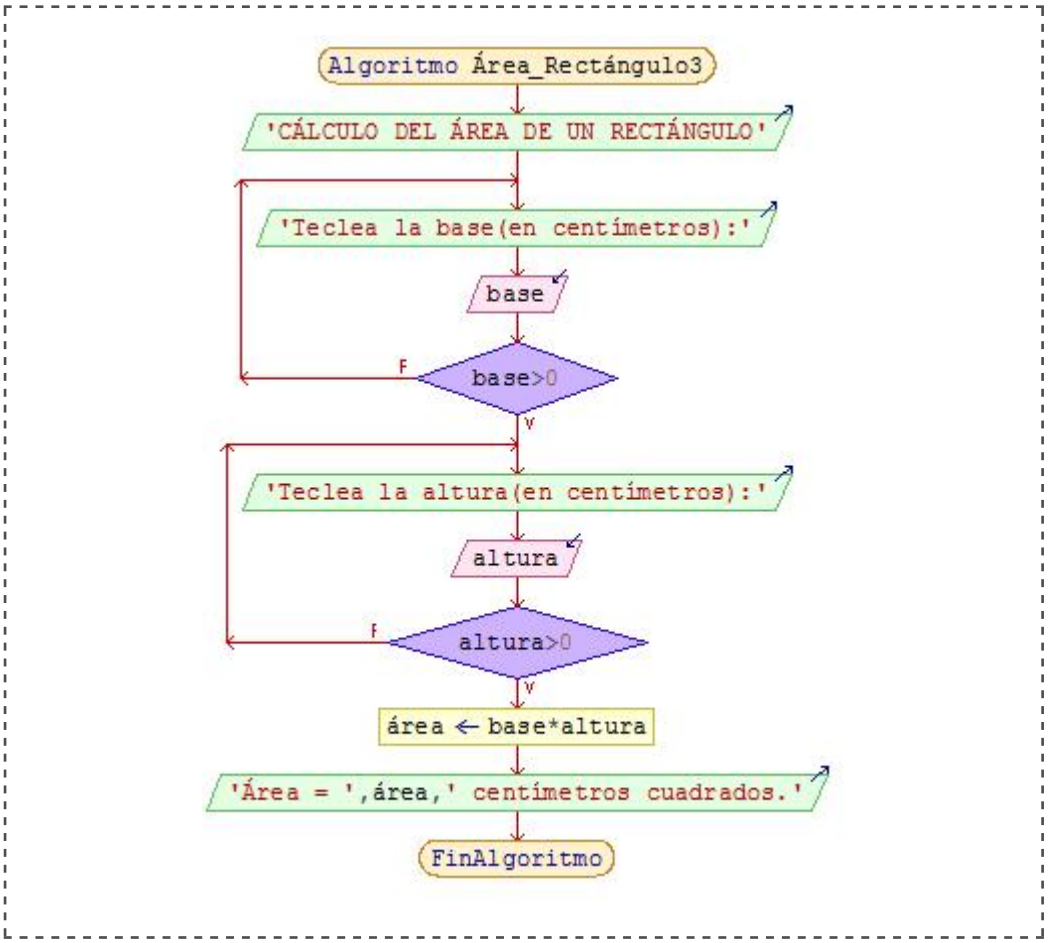
¿Cómo se haría exactamente? Escribiendo en el interior del rombo una expresión en la que se pregunte si la base es mayor que cero o no. Si la base no fuese mayor que cero el flujo del programa debería volver hacia atrás, justo hasta donde se vuelve a pedir al usuario la base del rectángulo. Si la base es mayor que cero el programa debería continuar su marcha de forma normal pidiendo la altura.



Análogamente, el problema se repetiría con la altura, no existen rectángulos con alturas negativas o con valor cero, pero eso ya no supone un problema importante para tí, ya que conoces la forma de solucionarlo... ¿Lo intentas tú mismo? ¡Ánimo!

Ejercicio resuelto

Mostrar retroalimentación



Para saber más

Esta estructura que acabas de utilizar para solventar el problema es conocida como **estructura repetitiva** o **bucle**, ya que permite repetir la ejecución de parte del algoritmo un número determinado de veces (en este caso hasta que la base sea mayor que cero, el programa seguirá pidiendo por teclado un valor positivo para dicha base). Existen diferentes tipos de estructuras repetitivas, fíjate en algunas de ellas aquí.

2.2.3 Ampliando el ejemplo

La ejecución del algoritmo de nuestro ejemplo, como ya sabes, pregunta al usuario la base y la altura para calcular el área del rectángulo con esas medidas, pero en cuanto se calcula, se muestra la solución y termina la ejecución del mismo.

Si el usuario necesitara calcular el área de otros rectángulos tendría que ejecutar de nuevo el programa, tantas veces como áreas tuviera que calcular.

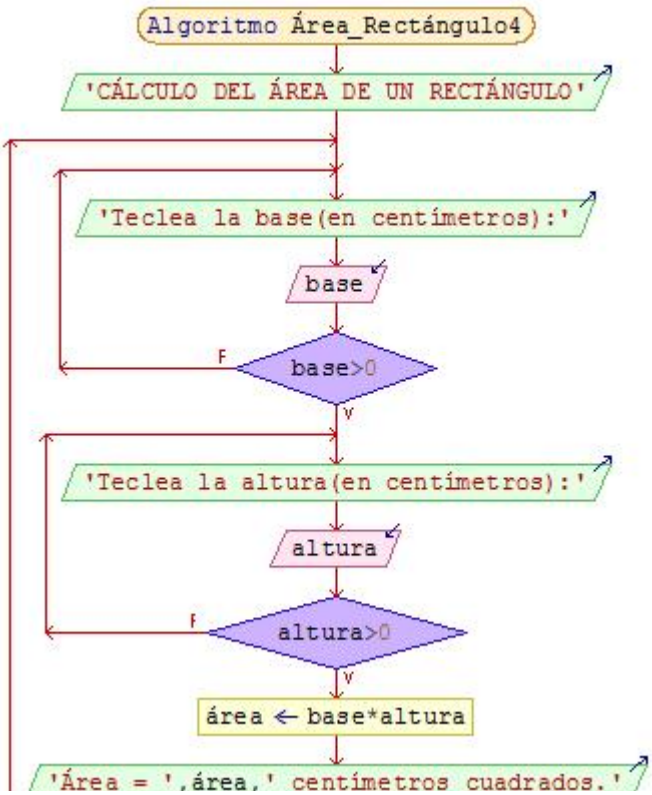
Sería interesante que el propio algoritmo, al finalizar el cálculo, preguntara al usuario si quiere calcular el área de otro rectángulo y volviese al principio para **seguir preguntando datos** y calculando, así hasta que el propio usuario no necesitara más cálculos.

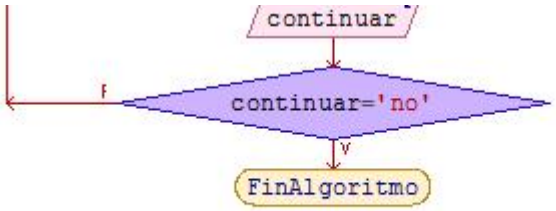
Reflexiona

¿Se te ocurre cómo modificar el algoritmo para conseguirlo? ¿Cómo lo plasmarías en el diagrama de flujo?...

Mostrar retroalimentación

... Efectivamente, de nuevo entra en juego un punto de decisión (el símbolo del rombo). Si justo después de calcular y mostrar el área, el programa le pregunta al usuario si desea continuar calculando áreas o no, se podría evaluar esa respuesta para hacer que el flujo del programa colocara la ejecución de nuevo al principio, o por el contrario, al responder que "no" el algoritmo finalizara su ejecución.





Importante

Las estructuras repetitivas (los bucles), y otras de control del flujo de ejecución, son potentes herramientas en programación, y como tales, merecen un desarrollo más extenso. No te preocupes, más adelante, en unidades posteriores, se verán en mayor profundidad.

2.3 Ventajas e inconvenientes

Imagen en Flickr de Imrishale con CC

Como todo en esta vida, la representación de algoritmos mediante el uso de los diagramas de flujo tiene ventajas, pero como podrás suponer, también posee algún que otro inconveniente.

La ventaja principal no es otra que la de mejorar la captación del proceso al mostrarlo como un dibujo (una imagen vale más que mil palabras...). Efectivamente, la persona que vea el diagrama, en un instante podrá hacerse una idea bastante aproximada de lo que el algoritmo pretende resolver. No es la única, ya que si se consigue un buen diagrama se puede reemplazar gran cantidad de texto explicativo. Además, en los diagramas de flujo se identifican de forma más intuitiva los pasos y los flujos de los datos.

No obstante, tiene un problema que puede llegar a ser importante: la superficie de representación necesaria a veces se queda muy pequeña y por tanto limita el diagrama. En efecto, cuando el algoritmo que se desea representar es muy extenso, se necesita "romper" el diagrama, necesitando hacerlo en exceso en multitud de ocasiones (ya viste que se procura dividir el diagrama en varios diagramas menores y se interconectan luego entre sí), haciendo por tanto el efecto contrario al que se pretende y convirtiendo la representación en un auténtico laberinto de conexiones.

No te preocupes, la siguiente herramienta de representación de algoritmos que verás, el pseudocódigo, solventa esta dificultad.

3. Pseudocódigo

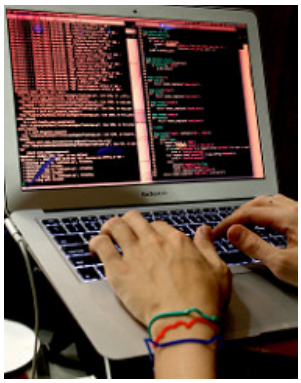


Imagen en Flickr de hackNY.org con CC

Es otro tipo de representación de algoritmos, en este caso, utilizando texto para describir las acciones y operaciones a realizar. Es un paso ya muy cercano a implementar el programa en un lenguaje de programación, pero con la ventaja de poderse utilizar texto en el lenguaje que se desee, por ejemplo en español, aunque el ordenador no lo entienda directamente. Así, se puede decir que el pseudocódigo consiste en representar los pasos de un algoritmo mediante narrativa adaptada a cualquier lenguaje entendible por las personas. El Pseudocódigo utiliza palabras que indican el proceso a realizar.

Entre las ventajas que existen de utilizar un pseudocódigo se puede destacar que permite representar en forma fácil operaciones repetitivas complejas. Además es muy fácil traducir pasar de pseudocódigo a programa en cualquier lenguaje de programación. Si se siguen las reglas se puede observar claramente la estructura que tiene cada operación. Para realizar un programa, sólo hay que desarrollar cada paso del pseudocódigo con las instrucciones propias del lenguaje de programación que se elija. Por tanto el pseudocódigo ofrece ya la solución al problema, mientras el lenguaje de programación se convierte en la herramienta con la que se muestra esa solución.

Generalmente, aunque existen excepciones, un pseudocódigo se divide en 3 fases: inicial, repetitiva o ciclo y final. En la primera se realizarán una serie de operaciones iniciales (abrir ficheros, inicializar variables, etc.), es decir, en general, todo lo que el ordenador debe hacer una sola vez. En la fase central, que es el cuerpo del programa, se repetirá una serie de acciones mientras se cumpla una o más condiciones, o hasta que dejen de cumplirse. A lo largo del programa se utilizarán diversos tipos de estructuras de control de varios tipos: secuenciales, condicionales y repetitivas. En la fase final concluiremos el programa, cerrando archivos, mostrando la solución por pantalla o impresora, etc.

Una característica importante del pseudocódigo es su cercanía a los lenguajes de programación, pero a la vez, su independencia de los mismos. Es por ello por lo que el pseudocódigo es utilizado en las etapas previas al desarrollo del software, empleándose a modo de boceto antes de proceder a la programación en cada lenguaje específico.

Importante

La representación de un algoritmo por tanto puede hacerse mediante diagrama de flujos y mediante pseudocódigo. En las fases iniciales de desarrollo del software, lo que se suele hacer es una primera representación con diagramas de flujo, para luego hacer la transformación del mismo a pseudocódigo. Por último, se traduce el pseudocódigo al lenguaje de programación a utilizar para la implementación definitiva.

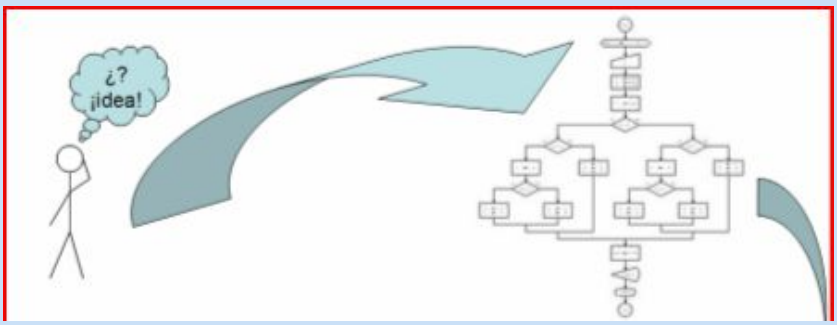


Imagen de creación propia bajo licencia CC

3.1 Elementos, sintaxis y convenciones

Para representar las acciones de los algoritmos mediante pseudocódigo, se necesita una serie de elementos para reflejar cada tipo de acción. Al usarse texto como herramienta básica en la representación, también se necesita una sintaxis concreta. Dicha sintaxis puede ser más estricta o menos, según se necesite en cada momento. Para primeros esbozos de la representación se podría optar por una sintaxis más burda, menos estricta, mientras que para los siguientes refinamientos se podría optar por utilizar ya una mayor precisión y por tanto una sintaxis más estricta. Además, también se le añade a la representación unas pocas convenciones que aportan una mayor claridad en el resultado final. Estas convenciones se mencionarán al final de este apartado.

En general, los elementos y su sintaxis podrían resumirse en la siguiente tabla:

SENTENCIAS/ ESTRUCTURAS	**PSEUDOCÓDIGO**	**DESCRIPCIÓN**
ASIGNACIÓN	←	Se utiliza para realizar una asignación de un valor a una variable. Por ejemplo, si se necesita asignar un valor 3 a la variable llamada coste, se escribiría: coste ← 3
OPERADORES ARITMÉTICOS	- (resta) + (suma) / (división) * (multiplicación) mod (resto de la división entera)	Se usan para formar expresiones en las que se realizarán operaciones aritméticas.
OPERADORES RELACIONALES	> (mayor que) >= (mayor o igual que) < (menor que) <= (menor o igual que) = (igual) <> (distinto)	Suelen utilizarse para formar condiciones que luego formarán parte de algunas estructuras de control o sentencias.
OPERADORES LÓGICOS	Y (and) O (or) NO (not)	Sirven para concatenar varias condiciones y formar una sola expresión con todas ellas.
ENTRADA / SALIDA	LEER nombre_variable ESCRIBIR expresión	La sentencia LEER es la que permite introducir valores por teclado (el ordenador, cuando llega a este tipo de sentencias se para y espera a que el usuario teclee un valor, luego sigue la ejecución del programa). La sentencia ESCRIBIR permite mostrar información por pantalla (u otro dispositivo de salida distinto)

ESTRUCTURA CONDICIONAL SIMPLE	SI (condición) ENTONCES Sentencias FINSI	Esta estructura permite ejecutar una serie de sentencias siempre y cuando se cumpla la condición que se desea evaluar. Si la condición se cumple, las sentencias que están en el interior de la estructura se ejecutan, si la condición no se cumple, el flujo del programa salta hasta la parte final de la estructura (FINSI) sin ejecutar las sentencias del interior.
ESTRUCTURA CONDICIONAL COMPUESTA	SI (condición) ENTONCES Sentencias SINO Otras Sentencias FINSI	Es una estructura parecida a la anterior, con la diferencia de que en este caso, si la condición evaluada no se cumple, el flujo del programa va directo al "SINO" y ejecuta todas las sentencias que encuentra entre el "SINO" y el "FINSI". Es decir, que dependiendo de si la condición se cumple o no, se ejecutan unas sentencias u otras.
ESTRUCTURA DE SELECCIÓN MÚLTIPLE	SEGÚN (variable) HACER CASO constante1: Sentencias ROMPER CASO constante2: Sentencias ROMPER CASO constante3: Sentencias ROMPER OTROS-CASOS: Sentencias FINSEGÚN	Esta estructura se parece a la anterior, pero en lugar de tener 2 caminos posibles de ejecución, tiene múltiples opciones. Se evalúa el valor de una variable y dependiendo de dicho valor, el flujo de ejecución salta al camino adecuado
ESTRUCTURAS REPETITIVAS	MIENTRAS (condición) HACER Sentencias FIN MIENTRAS REPETIR Sentencias	Estas estructuras son las llamadas "bucles". En definitiva lo que pretenden es ejecutar el grupo de sentencias que hay en su interior un número de veces (puede ir desde cero veces, una vez o más). En unidades posteriores se estudiarán con más detalle.

	HASTA QUE (condición) PARA variable←valor_inicial HASTA valor_final CON PASO valor_paso HACER Sentencias FINPARA	

Importante

Algo muy importante a la hora de escribir pseudocódigo, que también se extenderá al uso posterior de los lenguajes de programación (independientemente del lenguaje que sea): **EL USO DE LAS TABULACIONES AL ESCRIBIR SENTENCIAS QUE VAN DENTRO DE LAS ESTRUCTURAS DE CONTROL ES DE VITAL IMPORTANCIA PARA PODER INTERPRETAR BIEN EL ALGORITMO**. Por tanto, aunque dichas tabulaciones no son necesarias (porque los ordenadores las ignorarían), es muy conveniente y altamente recomendable, ya que las personas que tengan que estudiar el código fuente del programa (así se denomina al conjunto de instrucciones que componen el programa) tendrán mayor facilidad para entenderlo al ver las tabulaciones.

Si aún no no te convence esta afirmación prueba a contemplar un mismo trozo de código sin tabulaciones y con tabulaciones, y saldrás de dudas inmediatamente.

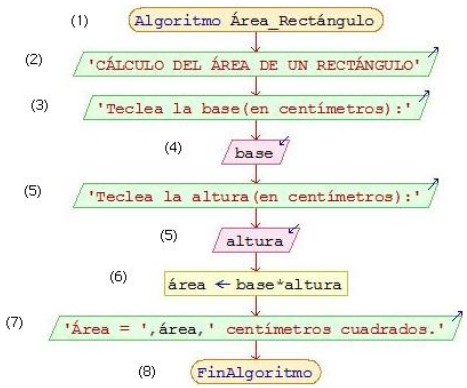
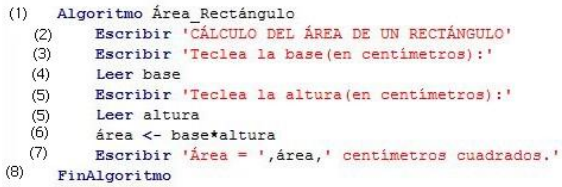
CÓDIGO SIN TABULACIONES	CÓDIGO CON TABULACIONES
LEER numero MIENTRAS numero<>0 HACER SI numero<0 ENTONCES ESCRIBIR numero*2 SINO ESCRIBIR numero/2 FINSI LEER numero FINMIENTRAS	LEER numero MIENTRAS numero<>0 HACER SI numero<0 ENTONCES ESCRIBIR numero*2 SINO ESCRIBIR numero/2 FINSI LEER numero FINMIENTRAS

Reflexiona sobre lo siguiente: si en un trozo de código tan pequeño, la diferencia de comprensión entre un modo y otro es enorme, imagina cuando la cantidad de código se multiplique...

3.2 Seguimos con el ejemplo

Ahora toca aplicar todos esos elementos para conseguir representar algoritmos. Para ello, continuarás con el ejemplo que ya conoces relativo al cálculo del área de un rectángulo. Podrás apreciar cómo quedaría la representación en pseudocódigo de todas las versiones de ese algoritmo.

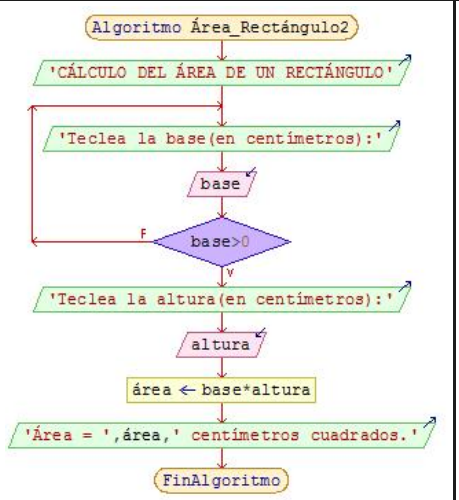
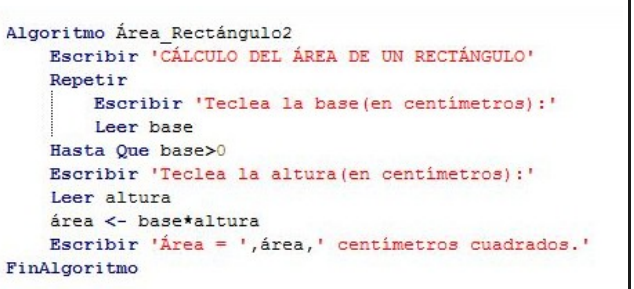
Versión 1: Realizar un algoritmo que pida por teclado la base y la altura de un rectángulo y devuelva por pantalla el área del mismo.

DIAGRAMA DE FLUJO	PSEUDOCÓDIGO
	


```
(1)   Algoritmo Área_Rectángulo
(2)       Escribir 'CÁLCULO DEL ÁREA DE UN RECTÁNGULO'
(3)       Escribir 'Teclea la base(en centímetros):'
(4)       Leer base
(5)       Escribir 'Teclea la altura(en centímetros):'
(5)       Leer altura
(6)       área <- base*altura
(7)       Escribir 'Área = ',área,' centímetros cuadrados.'
(8)   FinAlgoritmo
```

En este primer ejemplo de pseudocódigo se han numerado las líneas para hacerlas coincidir con la numeración que cada símbolo tiene asignada en el diagrama de flujo. Esta numeración, tanto en el diagrama de flujo como en el pseudocódigo es meramente didáctica, no es necesaria cuando se programa.

Versión 2: Idem pero repitiendo la petición por teclado de la base hasta que ésta sea mayor que cero.

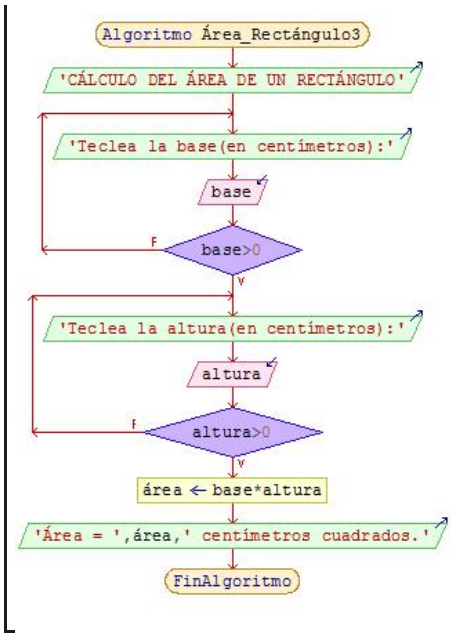
DIAGRAMA DE FLUJO	PSEUDOCÓDIGO
	


```
Algoritmo Área_Rectángulo2
    Escribir 'CÁLCULO DEL ÁREA DE UN RECTÁNGULO'
    Repetir
        Escribir 'Teclea la base(en centímetros):'
        Leer base
    Hasta Que base>0
    Escribir 'Teclea la altura(en centímetros):'
    Leer altura
    área <- base*altura
    Escribir 'Área = ',área,' centímetros cuadrados.'
FinAlgoritmo
```

Para dar mayor claridad en el entendimiento del pseudocódigo, a cada estructura que contiene más sentencias en su interior se le suele dibujar una línea vertical que va desde el comienzo de la estructura hasta el final de la misma (en el bucle REPETIR se puede apreciar).

Versión 3: Idéntico al anterior pero con el añadido de repetir la petición de la altura por teclado hasta que ésta sea mayor que cero.

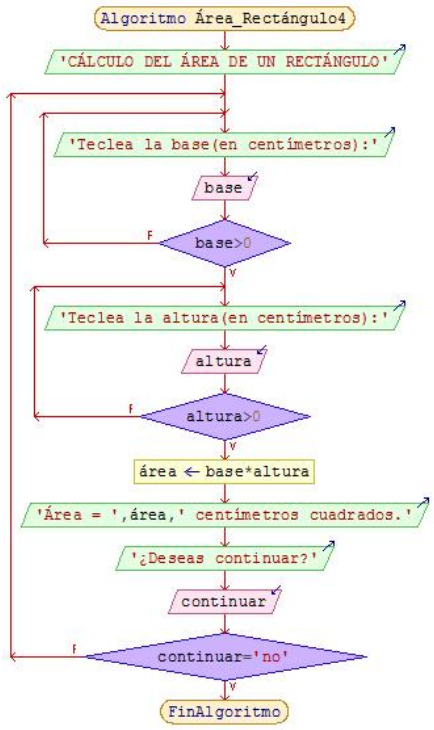
DIAGRAMA DE FLUJO	PSEUDOCÓDIGO




```
Algoritmo Área_Rectángulo3
    Escribir 'CÁLCULO DEL ÁREA DE UN RECTÁNGULO'
    Repetir
        Escribir 'Teclea la base(en centímetros):'
        Leer base
    Hasta Que base>0
    Repetir
        Escribir 'Teclea la altura(en centímetros):'
        Leer altura
    Hasta Que altura>0
    área <- base*altura
    Escribir 'Área = ',área,' centímetros cuadrados.'
FinAlgoritmo
```

Versión 4: Mejorar el programa anterior haciendo que se repita el cálculo del área de otro rectángulo hasta que el usuario responda que no desea continuar.

DIAGRAMA DE FLUJO	PSEUDOCÓDIGO




```
Algoritmo Área_Rectángulo4
    Escribir 'CÁLCULO DEL ÁREA DE UN RECTÁNGULO'
    Repetir
        Repetir
            Escribir 'Teclea la base(en centímetros):'
            Leer base
        Hasta Que base>0
        Repetir
            Escribir 'Teclea la altura(en centímetros):'
            Leer altura
        Hasta Que altura>0
        área <- base*altura
        Escribir 'Área = ',área,' centímetros cuadrados.'
        Escribir '¿Deseas continuar?'
        Leer continuar
    Hasta Que continuar='no'
FinAlgoritmo
```

4. Software para crear diagramas de flujo y pseudocódigo

En la actualidad existe una gran variedad de software, tanto para la elaboración de diagramas de flujo, como de pseudocódigo, unos son comerciales y otros de uso libre, incluso el software ofimático contiene herramientas que permiten dicho diseño.

- Para el diseño de diagramas de flujo puedes utilizar dos herramientas gratuitas: FreeDFD y el programa Dia.
- Para el desarrollo de pseudocódigo una muy buena opción es PSeInt.

Un poco más sobre PSeInt.

PSeInt es un entorno que cuenta con múltiples opciones y merece mención a parte.

Permite representar algoritmos utilizando tanto diagramas de flujo como pseudocódigo, incluso pasar de una forma a otra de manera inmediata.

Posibilita la ejecución del algoritmo como si éste ya estuviera implementado en un lenguaje de programación, con la opción, si se desea, de ir ejecutándolo paso a paso (esta manera de ejecutar un programa es muy utilizada cuando se intenta detectar el origen de algún problema).

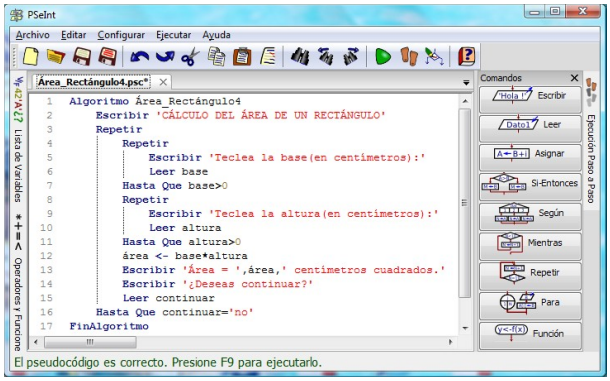


Imagen de creación propia bajo licencia CC

El editor ofrece distintos niveles de ayuda, lo que permite familiarizarse con la sintaxis de las instrucciones de una forma sencilla y rápida.

Otra característica muy interesante es la posibilidad de traducir el algoritmo a distintos lenguajes de programación, lo que constituye una ventaja enorme para los programadores que están iniciándose.

PSeInt es software libre, gratuito y multiplataforma y, por sus características, una excelente herramienta para todo programador. Descarga la última versión e intálala en tu equipo, para sí poder usarla a la hora de hacer tus primeros programas.

Para saber más

Más sobre PSeInt.

Para profundizar más sobre el uso de esta magnífica herramienta puedes acceder aquí.

Comprueba lo aprendido

Un diagrama de flujo:

- No constituye por si solo un algoritmo.
- Junto con el Pseudocódigo correspondiente conforman el algoritmo completamente.
- No puede tener un número infinito de sentencias.

Incorrecto

Incorrecto

Opción correcta

Solution

1. Incorrecto
2. Incorrecto
3. Opción correcta

Un lenguaje de programación:

- Posee características propias que condicionan al pseudocódigo.
- Pueden usarse distintos aplicados a un mismo algoritmo para resolver un problema.
- Es incompatible con el diagrama de flujo.

Incorrecto

Opción correcta

Incorrecto

Solution

1. Incorrecto
2. Opción correcta
3. Incorrecto

El ordenador:

Entiende los programas realizados derivados de sus algoritmos.

Incorrecto

Incorrecto

Opción correcta

Solution

1. Incorrecto
2. Incorrecto
3. Opción correcta

Un diagrama de flujo:

Debe de ser desarrollado con máximo detalle y textos.

No puede albergar ni un sólo texto.

Se construye con símbolos y la menor cantidad de textos posible.

Incorrecto

Incorrecto

Opción correcta

Solution

1. Incorrecto
2. Incorrecto
3. Opción correcta

En la construcción de un diagrama de flujo:

Los símbolos de inicio y fin son prescindibles.

No deben cruzarse las líneas de conexión entre secuencias.

No puede haber conectores.

Incorrecto

Opción correcta

Incorrecto

Solution

Comprueba lo aprendido

Contesta Verdadero o Falso a las siguientes cuestiones:

Diagramas de flujo y pseudocódigos son algoritmos.

Verdadero Falso

Verdadero

Dado un pseudocódigo concreto, determinará inequivocamente el tipo de lenguaje de programación a usar.

Verdadero Falso

Falso

Un lenguaje de programación conecta a personas y ordenadores.

Verdadero Falso

Verdadero

Un lenguaje de programación tiene su propia sintaxis.

Verdadero Falso

Verdadero

Un algoritmo puede tener un número infinito de pasos si el problema a solucionar es muy grande.

Verdadero Falso

Falso

Si un diagrama de flujo está mal diseñado puede ser corregido por un lenguaje de programación potente.

Falso

Cualquier símbolo usado en los diagramas de flujo puede soportar varias salidas hacia la sentencia siguiente.

Verdadero Falso

Falso

Los puntos de decisión de la acción son dirimidos por el propio lenguaje de programación, el usuario no actúa.

Verdadero Falso

Falso

6. ¿Te atreves?

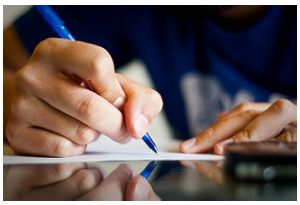


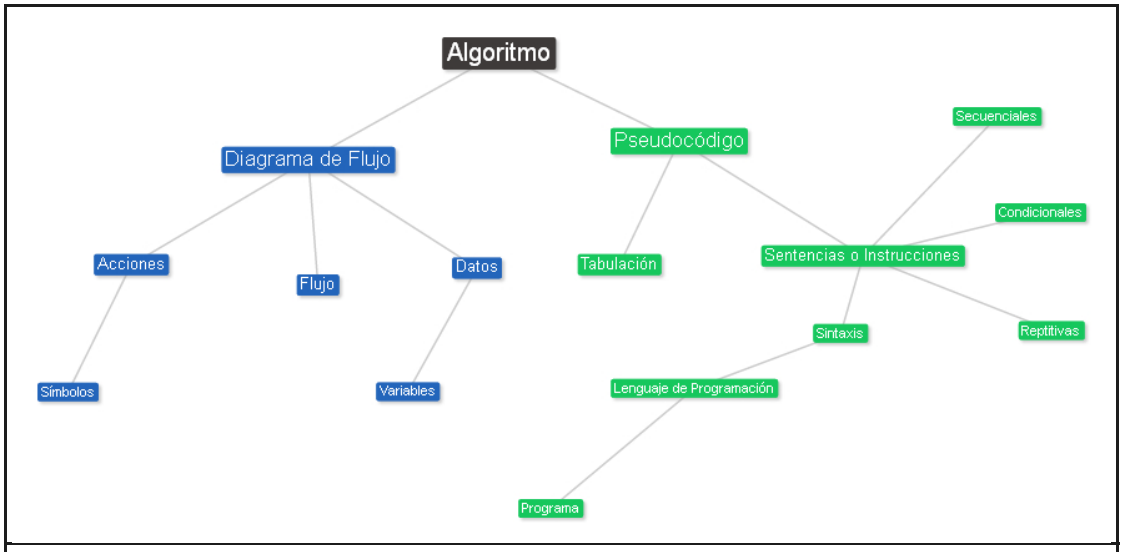
Imagen en Flickr de Lucas TheExperience con CC

¡La práctica hace milagros! Y en programación es fundamental, así que, ¿por qué no intentas tú mismo desarrollar algunos algoritmos representándolos en pseudocódigo?

Utiliza el entorno PSeInt para ello. Si encuentras dificultades puedes hacer un uso previo de los diagramas de flujo, por supuesto. ¡Ánimo!

Puedes empezar por... Realizar en pseudocódigo (o diagrama de flujo) los siguientes algoritmos:

a. Calcular el volumen de una esfera, una vez que se introduzca por teclado su radio.
b. Pedir por teclado el precio de un artículo en dos establecimientos distintos y calcular la media de ambos precios.
c. Construya todas las potencias de 2 para exponentes desde 1 a 10, y refleje sus valores.
d. Pedir por teclado dos números y calcular su suma, diferencia, multiplicación y división. ¿Qué ocurre si se divide por cero?...
e. Pedir por teclado un número del 1 al 12 y que el programa diga el nombre del mes.
f. Calcule la calificación de un exámen test de 20 preguntas, pidiéndonos los números de respuestas acertadas, falladas y en blanco, y sabiendo que las respuestas acertadas valen 0,5 puntos, las falladas restan 0,1 puntos, y las que están en blanco no puntúan.
g. Imprimir todos los números primos menores de 100.



Descargar PDF

Aviso Legal

El presente texto (en adelante, el "**Aviso Legal**") regula el acceso y el uso de los contenidos desde los que se enlaza. La utilización de estos contenidos atribuye la condición de usuario del mismo (en adelante, el "**Usuario**") e implica la aceptación plena y sin reservas de todas y cada una de las disposiciones incluidas en este Aviso Legal publicado en el momento de acceso al sitio web. Tal y como se explica más adelante, la autoría de estos materiales corresponde a un trabajo de la **Comunidad Autónoma Andaluza, Consejería de Educación y Deporte (en adelante Consejería de Educación y Deporte**).

Con el fin de mejorar las prestaciones de los contenidos ofrecidos, la Consejería de Educación y Deporte se reserva el derecho, en cualquier momento, de forma unilateral y sin previa notificación al usuario, a modificar, ampliar o suspender temporalmente la presentación, configuración especificaciones técnicas y servicios del sitio web que da soporte a los contenidos educativos objeto del presente Aviso Legal. En consecuencia, se recomienda al Usuario que lea atentamente el presente Aviso Legal en el momento que acceda al referido sitio web, ya que dicho Aviso puede ser modificado en cualquier momento, de conformidad con lo expuesto anteriormente.

Régimen de Propiedad Intelectual e Industrial sobre los contenidos del sitio web.

Imagen corporativa. Todas las marcas, logotipos o signos distintivos de cualquier clase relacionados con la imagen corporativa de la Consejería de Educación y Deporte que ofrece e