



UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

**Centro de
e-Learning**

FUNDAMENTOS DE LA PROGRAMACIÓN

Centro de e-Learning SCEU UTN - BA.

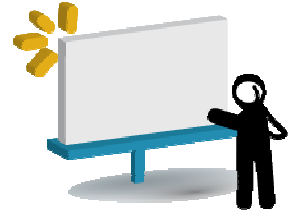
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148

www.sceu.frba.utn.edu.ar/e-learning



MÓDULO 1 - UNIDAD 4

Diagramas de flujo y Seudocódigo



Presentación:

En esta Unidad nos enfocaremos en la formas de representar algoritmos que usaremos en este y el próximo Módulo: diagramas de flujo y pseudocódigo.

Inicialmente nos enfocaremos en los formatos y convenciones de los diagramas de flujo para luego analizar un ejemplo resuelto de varias formas distintas, pero llegando siempre al mismo resultado.

Posteriormente, nos adentraremos otra forma de representar algoritmos: el pseudocódigo. Esta forma nos acompañará en el resto del curso.



Objetivos:

Que los participantes:

- Conozcan e incorporen las dos formas principales de representación de algoritmos que serán utilizados en este Módulo: diagramas de flujo y pseudocódigo
- Conozcan e incorporen las convenciones de los mismos
- Conozcan e incorporen los formatos de los diagramas de flujo y que los comprendan a través de un ejemplo práctico
- Conozcan e incorporen los aspectos principales de la representación de algoritmos usando pseudocódigo



Bloques temáticos:

1. Diagramas de flujo

1.1 Formatos

1.2 Convenciones

1.3 Ejercicio resuelto

2. Seudocódigo

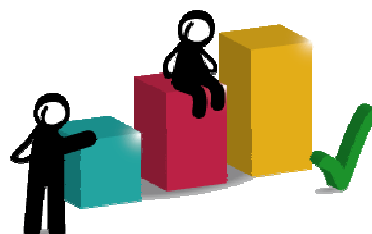
2.1 Convenciones

2.2 Documentación de código

2.3 Operadores aritméticos y relacionales

2.4 Desarrollo en pseudocódigo

2.5 Conclusiones



Consignas para el aprendizaje colaborativo

En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen algunas de las actividades sugeridas y compartan en los foros los resultados obtenidos.

Además, también se propondrán reflexiones, notas especiales y vinculaciones a bibliografía y sitios web.

El carácter constructivista y colaborativo del MEC nos exige que todas las actividades realizadas por los participantes sean compartidas en los foros.

* El MEC es el modelo de E-learning colaborativo de nuestro Centro.



Tomen nota

Las actividades son opcionales y pueden realizarse en forma individual, pero siempre es deseable que se las realice en equipo, con la finalidad de estimular y favorecer el trabajo colaborativo y el aprendizaje entre pares. Tenga en cuenta que, si bien las actividades son opcionales, su realización es de vital importancia para el logro de los objetivos de aprendizaje de esta instancia de formación. Si su tiempo no le permite realizar todas las actividades, por lo menos realice alguna, es fundamental que lo haga. Si cada uno de los participantes realiza alguna, el foro, que es una instancia clave en este tipo de cursos, tendrá una actividad muy enriquecedora.

Asimismo, también tengan en cuenta cuando trabajen en la Web, que en ella hay de todo, cosas excelentes, muy buenas, buenas, regulares, malas y muy malas. Por eso, es necesario aplicar filtros críticos para que las investigaciones y búsquedas se encaminen a la excelencia. Si tienen dudas con alguno de los datos recolectados, no dejen de consultar al profesor-tutor. También aprovechen en el foro proactivo las opiniones de sus compañeros de curso y colegas.



1. Diagramas de flujo

También conocidos como flujogramas o DFD (diagramas de flujo de datos), son representaciones gráficas o diagramas en los que se utilizan símbolos estándares para representar un proceso y la circulación de datos en el mismo, las operaciones que lo componen y la secuencia en que éstas se llevan a cabo.

Los símbolos aceptados como estándar para realizar un diagrama de flujo son los que se detallan a continuación:

- Terminal:



Indica el comienzo y finalización de un algoritmo. Es un rectángulo con bordes redondeados.

- Proceso:



Representa un proceso o una instrucción. Es un rectángulo.

- Decisión:



Indica un desvío en el flujo por toma de decisiones. Se representa con un rombo



- Entrada:



Utilizado para representar el ingreso de datos

- Salida:



Utilizado para representar una salida de datos

- Conector externo:



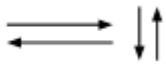
Indica una interconexión entre distintos diagramas

- Conector interno:



Indica una conexión dentro del mismo diagrama

- Línea de flujo:



Conecta símbolos e indica el flujo del diagrama

Se utilizan normalmente para modelar rutinas simples y para representar distintos tipos de procesos. Si bien su uso para representar algoritmos de programación cayó en cierto desuso, es muy común ver este tipo de diagramas siendo utilizados para modelar procesos administrativos en manuales de procedimientos o de calidad, aunque puede resultar de gran utilidad para aquellos que se inician en la programación, dada su simplicidad de uso y el hecho de brindar una vista familiar o más amigable a un algoritmo.



Otras de las ventajas que proporciona la utilización de diagramas de flujo es la rápida visualización de las actividades que pudieran ser innecesarias en un proceso, a la vez que es posible verificar si la distribución del trabajo está equilibrada en el proceso.

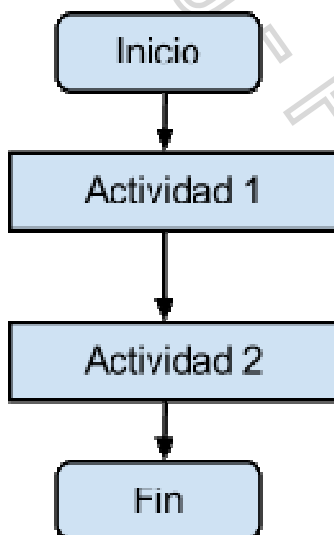
Resumiendo, podemos decir que las principales características son:

- Poder de síntesis
- Notación estándar
- Amplia difusión y aceptación
- Fácil comprensión
- Se elabora con sencillez y no requiere técnicas ni recursos complejos.

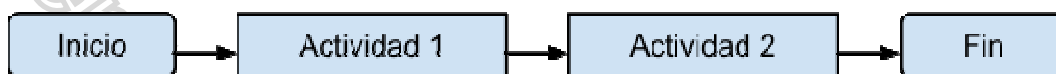
1.1 Formatos

Los diagramas de flujo pueden ser realizados con diversos formatos, como ser:

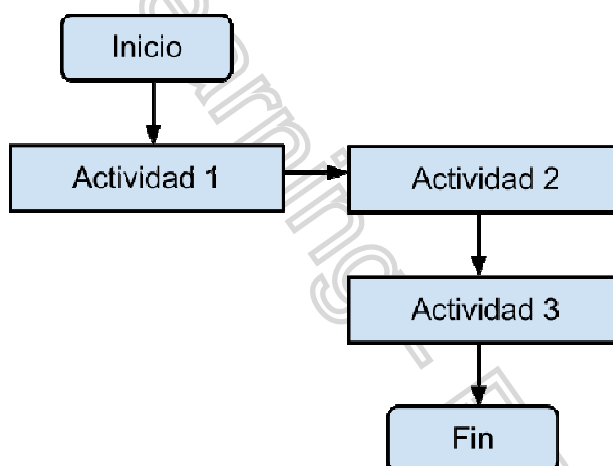
- **Vertical:** el flujo se inicia desde la parte superior y avanza hacia abajo. Por ejemplo:



- **Horizontal:** la secuencia de operaciones se desarrolla de izquierda a derecha. Por ejemplo:



- **Panorámico:** se trata de una combinación de los dos formatos anteriores, algunas veces agregando columnas al plano sobre las cuales el flujo se desarrolla. Estas columnas representarían el “espacio” (dicho en forma figurada) o área de competencia por el cual se ejecuta cada porción del algoritmo. Por ejemplo:



El primero es el de mayor difusión y es el que será el utilizado en el presente curso.

1.2 Convenciones

Algunas convenciones o buenas prácticas para construir diagramas de flujo son:



- Identificar con un nombre representativo el proceso o programa que se está desarrollando, evitando nombres del tipo “Programa1”, etc. Un ejemplo en el caso de estar diagramando un proceso de consulta de compras podrías ser “ConsultaCompras” o “Consulta_de_compras” o similar.
Deben estar claramente identificados y nombrados los puntos iniciales y finales del proceso.
- Representar los símbolos con un tamaño similar entre sí, aunque se prioriza la claridad, por lo que esta convención puede modificarse usando el buen juicio y el sentido común (si no es claro para el autor, menos lo será para otro que reciba el diagrama).
- Cada paso del diagrama debe ser una instrucción atómica y auto explicativa. Por ejemplo, un rectángulo que representa una actividad deberá contener una única instrucción o un conjunto de instrucciones de similares características.
- Tener en cuenta que el orden de ocurrencia de los pasos se representa por el orden en que aparecen los símbolos, de arriba hacia abajo.
- En el caso de estar diagramando un flujo para fines formales (por ejemplo, como parte de la documentación de la aplicación), se deberá documentar formalmente, agregando nombre del autor, fecha, versión del documento y una descripción del proceso o programa que se está modelando.
- Cada paso puede (no es requerimiento excluyente) identificarse con un número y agregar una pequeña descripción de cada uno.
- La redacción del contenido del símbolo debe ser realizada con frases breves y sencillas y estar escritas comenzando con un verbo en infinitivo (“Sumar...”, “Mostrar...”, “Ingresar...”, etc.)



- Evitar usar siglas que no se encuentren aclaradas en el mismo documento o en un documento de soporte anexo.
- Los símbolos de conectores pueden ser alfabéticos o numéricos, pero deben coincidir los conectores de entrada y salida; cuando existe una gran cantidad de conectores, es aconsejable asociar un color al símbolo.
- El diagrama debe dar una imagen de limpieza, orden, control y claridad.

1.3 Ejercicio resuelto

A continuación, desarrollaremos un ejemplo de diagrama de flujo simple (desde el punto de vista del problema y de la estructura con la que serán resueltos – que probablemente no sea la óptima, aunque sí válida).

Justamente por la simplicidad de los ejemplos evitaremos realizar el proceso de análisis de “Entrada-Proceso-Salida”, aunque instamos a realizarlos cada vez que sea posible y justificable.

PROBLEMA: Dado un número inicial, mostrar los siguientes 3 números naturales inmediatamente sucesivos.

A continuación veremos 3 posibles soluciones algorítmicas igualmente válidas para el problema planteado.

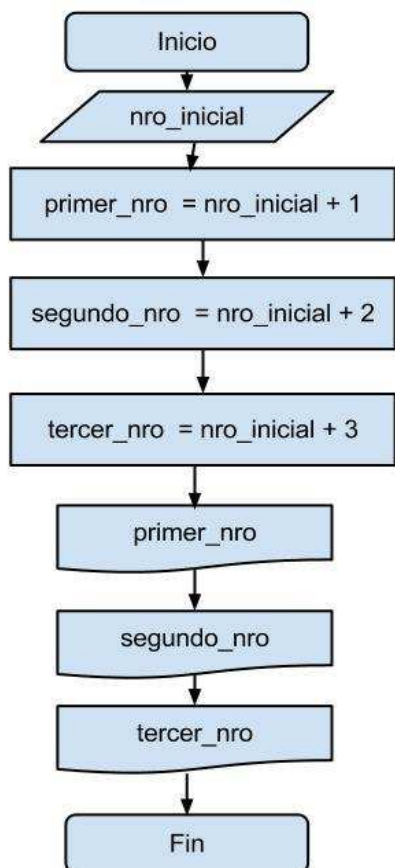
Esto nos deja entrever una de las tantas máximas que rigen el mundo de la programación:

“Para un problema dado, existen innumerables alternativas de solución”



Recomendamos analizar las 3 alternativas y posteriormente ver las explicaciones de cada una de ellas.

A)



En este caso, el algoritmo recibe un valor inicial, que se guarda en el elemento "nro_inicial". Este elemento (llamado "variable") se puede tomar figurativamente como una "bolsa" donde se ingresan distintos tipos de valores: en este caso, el número sobre el cual se debe construir la serie.

Posteriormente, se definen 3 nuevas variables (nuestras famosas "bolsas") donde se guarda el número inicialmente ingresado, al cual se le suman 1, 2 y 3, para obtener los siguientes valores que forman la serie.

En última instancia, se muestran los 3 valores y finaliza el algoritmo.

Un caso de prueba sencillo sería ingresar 2 valores (uno por cada vez que se ejecuta el algoritmo): primero 5 y luego 99, para tener un rango amplio de valores.



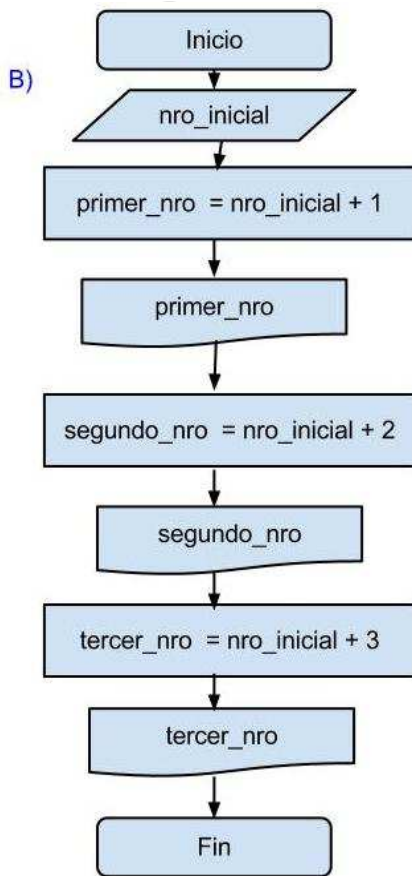
Repasemos el algoritmo para el primer caso de prueba:

1. Se ingresa 5, por lo que $nro_inicial = 5$
2. Luego, se le suma 1 a $nro_inicial$ (que vale siempre 5) y se lo guarda en una variable. Por lo que $primer_nro = 5 + 1 = 6$
3. Se le suma 2 a $nro_inicial$ y se lo guarda en otra variable. Por lo que $segundo_nro = 5 + 2 = 7$
4. Se le suma 3 a $nro_inicial$ y se lo guarda en una tercera variable. Por lo que $tercer_nro = 5 + 3 = 8$
5. Luego se muestran los valores de cada una de las variables:
 - a. $primer_nro = 6$
 - b. $segundo_nro = 7$
 - c. $tercer_nro = 8$

Repetimos la ejecución del caso de prueba con otro valor a probar:

1. Se ingresa 99
2. A 99 se le suma 1 ($primer_nro=100$)
3. A 99 se le suma 2 ($segundo_nro=101$)
4. A 99 se le suma 3 ($tercer_nro=102$)
5. Se muestra
 - a. 100
 - b. 101
 - c. 102

El algoritmo cumple correctamente con ambos casos de prueba, por lo que lo damos por satisfactorio.



Pasemos a analizar la segunda alternativa de solución.

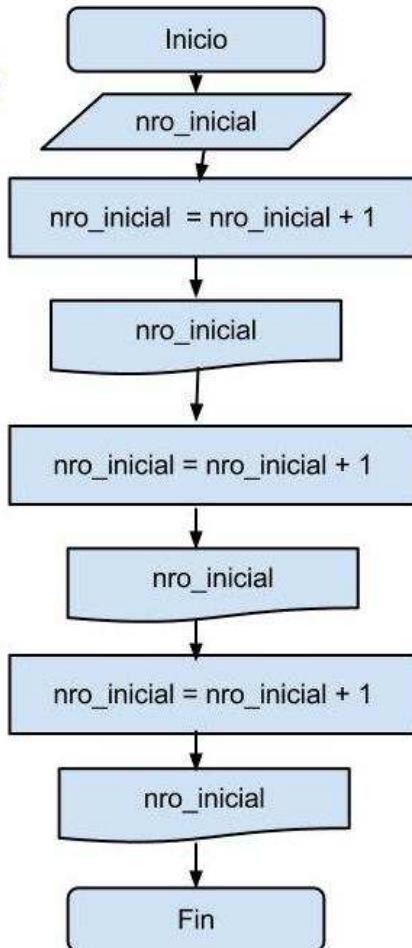
En este caso, la lógica del planteo es el mismo que en la alternativa de solución A).

La diferencia del planteo radica en el orden de los pasos de ejecución. En el primer caso, primero se realizaban todas las sumas y finalmente se mostraban todos los resultados, y en este caso a medida que se van calculando los números de la serie, se muestra cada uno de los resultados parciales.

El resultado final es idéntico en ambos, sumado a que el planteo también es similar.

Si ejecutamos los casos de prueba, comprobaremos que nuevamente se cumple con la premisa del problema planteado.

c)



Para finalizar con el presente ejemplo, pasamos a analizar la tercera alternativa de solución.

Este algoritmo presenta una importante diferencia con respecto a los 2 primeros. Es similar al segundo, ya que a medida que va calculando los números de la serie, va mostrando los resultados.

La diferencia radica en el manejo del incremento del valor inicial ingresado. Vemos que no existen las variables intermedias (primer_nro, segundo_nro y tercer_nro), sino que todas las operaciones se realizan sobre nro_inicial. Por esto es que no se suman los valores 1, 2 y 3, sino que es el mismo nro_inicial el que se va modificando (y mostrando).

En este algoritmo no podríamos mostrar los 3 resultados parciales al final, ya que estos estados intermedios se van pisando a medida que el algoritmo avanza.

Nos detenemos para validar el algoritmo, a través de la ejecución de un caso de prueba. En este caso haremos una única ejecución de prueba ingresando el valor 12.



1. Se ingresa 12, por lo que `nro_inicial = 12`
2. Luego, se le suma 1 a `nro_inicial` que pasa a valer 13
3. Se muestra este valor: 13
4. Se le suma 1 a `nro_inicial` que valía 13, por lo que pasa a valer 14
5. Se muestra este valor: 14
6. Se le suma 1 a `nro_inicial` que valía 14, por lo que pasa a valer 15
7. Se muestra este valor: 15

Vemos que los valores mostrados por el programa fueron 13, 14 y 15, por lo que damos el algoritmo por válido.

Más allá de la utilización de los diagramas de flujo, es importante tener en cuenta lo visto en este ejemplo: para un caso problema sencillo, rápidamente identificamos 3 soluciones posibles válidas. Queda claro la multiplicidad de alternativas de solución, lo cual suele potenciarse a medida que los problemas planteados aumentan en complejidad.

La pregunta en este caso podría ser: ¿cuál es la mejor solución? Si bien existen ciertos parámetros que nos pueden guiar para seleccionar una de las tres como la más óptima, esta clase de juicios suele ir refinándose con el tiempo y la práctica. En complemento (claro está) del buen juicio, sentido común y mejores prácticas que direccionen una tipología de problemas en particular.

Vale aclarar que algunos de los símbolos presentados en esta Unidad no fueron utilizados en los ejemplos dados. Esto se debe a que serán analizados con más detalle en el próximo Módulo.



ACTIVIDAD 1:

Suponiendo que la afirmación:

“Para un problema dado, existen innumerables alternativas de solución”

Es realmente cierta: ¿Cuál o cuáles serían los criterios para decidir que una alternativa de solución es mejor que otra?

Comparta sus respuestas en el foro de actividades de la Unidad.



2. Seudocódigo

Elseudocódigo, cuyo significado literal es “falso lenguaje”, es el tipo de descripción de alto nivel para representar algoritmos, sobre el que se encuentra basado el presente Curso, junto a los diagrama de flujo. De allí la importancia que le daremos a este tópico en la presente Unidad y en las siguientes. Esta importancia hace necesario definir el uso que le daremos alseudocódigo, ya que, al no haber un estándar unilateralmente aceptado, es posible encontrar numerosas formas de representarlo. Por lo que, a continuación, daremos algunos parámetros (que iremos enriqueciendo) para formalizar su uso en este Curso.

Como nombramos anteriormente, elseudocódigo es un subconjunto del lenguaje natural con algunas estructuras sintácticas propias que lo asemejan a los lenguajes de programación, como ser asignaciones, ciclos, condicionales y otros tipos de estructuras. Si bien su uso actual suele estar limitado al ámbito académico como base para teórico-práctica que permita una rápida adaptación a lenguajes de programación concretos, también suele estar presente en documentos de investigación y divulgación en los que resulta importante abstraerse de una implementación (lenguaje) en particular. También suele ser de gran ayuda en varias instancias en proyectos de desarrollo de software, por ejemplo, cuando surge la necesidad de validar con un usuario o actor no técnico un algoritmo, permitiendo mostrarlo en forma genérica sin la necesidad de que aquella persona conozca el lenguaje de programación en el cual finalmente será implementado.

Para esto, resulta necesario presentar unseudocódigo ordenado, limpio y claro, con el fin de que sea totalmente comprensible en forma inmediata. Por ello, nos vemos obligados a seguir algunas reglas o “mejores prácticas” que analizaremos a continuación.





2.1 Convenciones

Nos centraremos en definir los elementos que utilizaremos en nuestros primerosseudocódigos y posteriormente daremos algunos parámetros generales que deberán cumplir la estructura de todos losseudocódigos que realicemos.



Así como los diagramas de flujo tienen sus símbolos, definiremos algunas palabras reservadas haciendo una comparación con dichos símbolos. Inicialmente nos concentraremos en los que vamos a utilizar en esta Unidad, para ir incorporando el resto en el transcurso del Curso.

Por “palabra reservada” nos referimos a todos aquellos términos especiales que utilizaremos para definir las instrucciones que son propias del lenguaje, en este caso, del pseudocódigo.

Diagrama de Flujo		Seudocódigo
Símbolo	Nombre	Descripción
	Terminal	Se representa con las palabras reservadas “inicio” y “fin”, según el caso.
	Proceso	Se trata de toda instrucción que forme parte de un algoritmo que no represente entradas o salidas, pero sí un procesamiento de datos. Por ejemplo: una operación matemática. No lleva asociada ninguna palabra reservada.
	Entrada	Se representa con la palabra reservada “ingresar”. La sintaxis sería “ingresar:” + entrada de datos
	Salida	Se representa con la palabra reservada “mostrar”. La sintaxis sería “mostrar:” + salida de datos e información. En el caso de querer mostrar un mensaje, este deberá estar entre comillas dobles, por ejemplo: mostrar: “el resultado es:” mostrar: valorDelCalculo

Siempre escribiremos en minúscula, salvo para un caso puntual que definiremos más adelante.



En caso de tener nombres de programas o instrucciones que involucren más de una palabra (nombre compuesto), utilizaremos un patrón de escritura conocido como “lower camel case”. Este es un derivado del término “camel case” en inglés y que define que una sentencia (más de una palabra) se unifique en una palabra compuesta en la que se unen todas las palabras de la sentencia, aunque poniendo la primera letra de cada una en mayúscula. Por ejemplo para “mi programa de suma” usaremos la notación “MiProgramaDeSuma”. Esta sería la representación en “camel case”; sin embargo, nosotros usaremos la variación “lower camel case”, por lo que la primera letra será siempre minúscula. Entonces, en nuestro caso “mi programa de suma” será llamado “miProgramaDeSuma”.

Con respecto a la estructura, definiremos que todos nuestros programas deberán respetar el siguiente formato:

```
programa <<nombre del programa>>
```

```
inicio
```

```
    instrucción 1
```

```
    instrucción 2
```

```
    instrucción 3
```

```
    ...
```

```
    instrucción N
```

```
fin
```

Cuando programamos (ya sea con pseudocódigo o en un lenguaje de programación determinado) debemos tener en cuenta que cada instrucción va en una única línea de código y que las operaciones se deben escribir de la forma más clara que sea posible, para permitir que cualquier programador (o no) las entienda sin mayor esfuerzo. Es lo que se conoce como “claridad de código” y es una de las prácticas más importantes de la programación.



Hay que tener siempre en cuenta que una porción de código que creamos (o modificamos) nunca sabemos cuántas otras veces será visto y modificado por nosotros mismo o por otra persona. Por lo que generar un código legible siempre debe tomarse como una inversión, en lugar de una pérdida de tiempo.

De la estructura definida arriba, podemos rescatar un punto muy importante que hace a la claridad del código escrito: la indentación. Este término (se trata de un anglicismo) hace referencia a los espacios dejados entre el margen izquierdo en las líneas que contiene la "instrucción 1", "instrucción 2", etc. Este conjunto de espacios en blanco (también conocido como "tab", haciendo referencia a esa tecla en el teclado de una PC), permite identificar los distintos "niveles" que tiene el código. Que las 4 instrucciones mostradas arriba tengan "1 tab" de corrimiento con respecto al margen, permite ver como éstas son agrupadas dentro de un bloque o nivel que comienza en "inicio" y finaliza en "fin".

En la misma sintonía tenemos el siguiente punto planteado a continuación.

2.2 Documentación de código

La programación (como se dijo anteriormente) es sólo una parte dentro de un proyecto de desarrollo, complementado con otras actividades entre las cuales se encuentran la generación de la documentación del proyecto.

En relación a esta documentación, se encuentra un tipo de documento que surge directamente desde el programador, que está orientada a explicar y clarificar los aspectos más técnicos de la programación en el proyecto.

Esto es a lo que nos referimos con "documentación de código": el hecho de agregar comentarios al código fuente que estamos programando con la finalidad de explicar un algoritmo complejo, aclarar qué modificaciones se fueron introduciendo, quién es el autor del programa, la fecha en que fue creado o modificado, etc.

Para incluir esta clase de aclaraciones en el código, usaremos el operador "//" seguido del comentario, repitiendo la operación por cada línea a la cual se la quiera comentar.



Como mínimo y regla para este curso deberemos comentar (documentar) los siguientes puntos:

- Cabecera del programa:
 - Autor del programa
 - Fecha de creación
 - Descripción general del programa
- Cuerpo del programa
- Todas aquellas instrucciones complejas o que no resulten intuitivas

Un ejemplo podría ser:

```
//AUTOR: Emilio Rasic  
  
//FECHA: 24/04/2012  
  
//DESCRIPCION: este es un ejemplo de comentario general del programa  
// que utilizaremos en el curso, mostrado en dos líneas  
  
programa nombreDelPrograma  
  
inicio  
  instrucción 1    // este es un tipo de comentario en línea  
                  // que explica una determinada instrucción  
  
  instrucción 2  
  
  // este tipo de comentario se puede usar para explicar el bloque  
  // de código que viene debajo  
  
  instrucción 3  
  
  instrucción N  
fin
```




ACTIVIDAD 2:

Investigar y fundamentar por qué es importante "documentar" el código: ¿Quién lo hace? ¿Cuándo se hace: durante la programación o una vez que se finaliza? ¿Es realmente una ayuda? ¿A quién iría dirigida?

Comparta sus respuestas en el foro de actividades de la Unidad.



2.3 Operadores aritméticos y relacionales

El pseudocódigo permite la utilización de un conjunto de operadores aritméticos y relacionales que pueden ser utilizados para la construcción de las instrucciones necesarias.

A continuación, un detalle de los más usados:

	Operador	Significado	Ejemplo de uso
Relacionales	>	Mayor que	10>9
	<	Menor que	3<5
	= / <> o !=	Igual que / Distinto de	1=1 / 1<>2 o 1!=2
	>=	Mayor o igual que	9>=7
	<=	Menor o igual que	3<=7
Aritméticos	+	Suma	6+3
	-	Resta	4-2
	*	Multiplicación	3*5
	/	División	10/4
	^	Potencia	4^2
	%	Resto de división	6 % 3

La jerarquía de los operadores matemáticos es la misma a la del álgebra, aunque a través del uso del paréntesis puede modificarse.



2.4 Desarrollo en pseudocódigo

A continuación, desarrollaremos nuestro primer programa en pseudocódigo. Para esto, tomaremos como referencia el ejemplo visto anteriormente que fuera representado con un diagrama de flujo. Más precisamente, la solución A).

Comenzaremos por documentar (acción también conocida como “comentar”, referido al hecho de agregar comentarios documentales al código) el programa. Posteriormente recrearemos el algoritmo en el cuerpo del programa.

```
// AUTOR: Emilio Rasic  
// FECHA: 24/03/2012  
// DESCRIPCION: este programa muestra los 3 primeros números consecutivos a  
// partir de un número que es ingresado por el usuario  
  
programa serieDeTresNumeros  
inicio  
(sigue....)
```



(continúa...)

```
ingresar: nroInicial           // se ingresa el primer número (*)  
  
// a continuación se realizan las sumas para obtener los números de la serie (**)  
  
primerNro = nroInicial + 1  
  
segundoNro = nroInicial + 2  
  
tercerNro = nroInicial + 3  
  
// se muestran los 3 números de la serie (**)  
  
mostrar: primerNro  
  
mostrar: segundoNro  
  
mostrar: tercerNro
```

fin

No nos detendremos en la solución, sino que más bien repasaremos los elementos del programa para identificarlos claramente:

- El programa tiene un encabezado con un comentario inicial (autor, fecha y descripción del programa)
- El programa tiene un nombre descriptivo, escrito en “lowerCamelCase”
- El programa tiene un bloque principal, definido entre las palabras reservadas “inicio” y “fin”



- El contenido del bloque principal del programa está indentado, lo que permite ver la estructura del programa
- El contenido del bloque principal del programa tiene comentarios “en línea” y “por grupo de líneas”, refiriéndonos al primero como el que tiene un (*) y al segundo con los que tienen un (**)

2.5 Conclusiones

El pseudocódigo nos permite:

- Identificar inequívocamente cuál es el comienzo del programa
- Identificar claramente todos los puntos de finalización del programa
- Definir un número finito de instrucciones, estados intermedios y caminos entre el inicio y el final o las finalizaciones del programa
- Visualizar claramente los niveles de las estructuras del programa, a través de la indentación
- Comprensión de algoritmos por parte de personas que no tengan necesariamente conocimientos de lenguajes de programación
- Se encuentra basado en el lenguaje natural en el idioma de cada uno, por lo que se evitan palabras complejas o notaciones que no se interpreten inmediatamente
- Al ser un subconjunto del lenguaje natural, se evitan errores gramaticales, abreviaciones y puntuaciones que podrían dificultar la comprensión.



Actividades de la Unidad 4

- Responder a las actividades presentadas en esta unidad en el foro de actividades
- Analizar los ejercicios resueltos presentados en el Campus como material adicional.

¡¡ NO OLVIDEN REALIZAR LA EVALUACIÓN DEL MÓDULO 1 !!



Lo que vimos

- Formas principales de representación de algoritmos que serán utilizados en este Módulo: diagramas de flujo y pseudocódigo
- Convenciones de los mismos
- Formatos de los diagramas de flujo y que los comprendan a través de un ejemplo práctico
- Aspectos principales de la representación de algoritmos usando pseudocódigo



Lo que viene:

- Principales aspectos del paradigma estructurado de programación
- Estructuras de control y sus principales variantes
- Otras estructuras comunes

