UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

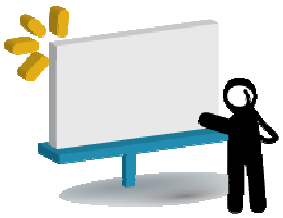
Centro de e-Learning

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

Pseudocódigo



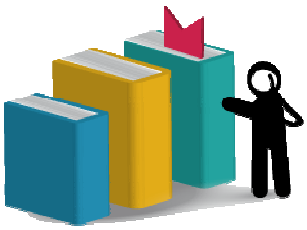
Presentación:

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

Objetivos:

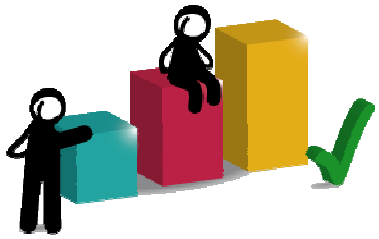
Que los participantes:

- Conozcan e incorporen la forma de representación de algoritmos que serán utilizados en este Módulo: el pseudocódigo
- Conozcan e incorporen las convenciones de los mismos
- Conozcan e incorporen los aspectos principales de la representación de algoritmos usando pseudocódigo



Bloques temáticos:

1. Pseudocódigo
    - 1.1 Convenciones
    - 1.2 Documentación de código
    - 1.3 Operadores aritméticos y relacionales
    - 1.4 Desarrollo en pseudocódigo
    - 1.5 Conclusiones
2. Ejemplo integrador
3. Tips sobre el desarrollo en pseudocódigo
4. Programas resueltos



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:

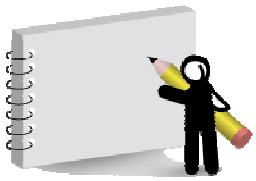
- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

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. Pseudocódigo

El pseudocódigo, 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. 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 al pseudocódigo, 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, el pseudocódigo 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.

En algunos casos estos se conoce como "meta lenguaje" al utilizar palabras y formulaciones propias del "negocio" o del ámbito en que se encuentren las personas que requieran validar una solución o comprender un problema en forma previa a la programación.

Para que este proceso de construcción sea efectivo, resulta necesario presentar un pseudocódigo 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.

1.1 Convenciones

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

Definiremos algunas "palabras reservadas" que tendrán una función específica. 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.

Pseudocódigo	
Palabra reservada	**Descripción**
"programa NOMBRE"	Nos indica que comienzo del programa. Siempre va acompañado del nombre del programa, que debe ser representativo de lo que hace
"inicio" / "fin"	Bloque principal donde vamos a ubicar nuestro código. Solo hay 1 "inicio" y solo hay 1 "fin" en todos los programas. El "inicio" siempre va luego del "programa..." y el "fin" es siempre la última línea de todo programa
"mostrar:"	Representa una salida en formato de texto del programa. Se pueden imprimir mensajes (siempre entre "") o valores de variable (tema que comentaremos un poco más adelante) La sintaxis es “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

"ingresar:"	Sirve para simular la acción en la cual un supuesto usuario de nuestro programa (la personal que lo "ejecutaría") ingresar un valor por teclado y ese valor queda guardo en una variable La sintaxis sería “ingresar:” + entrada de datos El caso de querer guardar un nombre de una persona en una variable sería por ejemplo: ingresar: nombreDeLaPersona

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” o "upper camel case" y lo vamos a utilizar para nombrar a los programas.

Para las variables, sin embargo, usaremos la variación “lower camel case”, por lo que la primera letra será siempre minúscula. Entonces, en nuestro caso “nombre de la persona” será llamado “nombreDeLaPersona”, empezando con minúscula.

Definidas las palabras reservadas, veremos cómo éstas se combinan entre sí para formar un programa.

Con respecto a la estructura, definiremos que todos nuestros programas deberán respeta 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 "sangría", "tabulado" o “tabs”, 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.

1.2 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 en forma narrativa, ad hoc, 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, o cualquier dato relevante que sea considere imprescindible para comprender un bloque de código.

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: 9/12/2018
//DESCRIPCION: este es un ejemplo de comentario general del programa
  //que utilizaremos en el curso, pero dividido en dos líneas

programa NombreDelPrograma
inicio
    instruccion 1       //este es un tipo de comentario en línea
                           //que explica una determinada instrucción
    instruccion 2
    //este tipo de comentario se puede usar para explicar el bloque
    //de código que viene debajo
    instruccion 3
    instruccion N
fin
```

ACTIVIDAD 1:

Investigar y fundamentar porqué es importante "documentar" el código: ¿Quién lo hace? ¿Cuándo se hace: durante la programación, antes 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.

1.3 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
	=	Igual que	1=1
	<> o !=	Distinto de	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.

1.4 Desarrollo en pseudocódigo

A continuación, desarrollaremos nuestro primer programa en pseudocódigo.

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: 16/01/2019
//DESCRIPCION: este programa muestra los 3 primeros números consecutivos a
    //a partir de un número que es ingresado por el usuario

programa SerieDeTresNumeros
inicio
    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 “UpperCamelCase”
- 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 (**)

1.5 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.

2. Ejemplo integrador

A continuación, desarrollaremos un ejemplo 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.

Versión 1:

```
programa MostrarNumerosV1
inicio
    ingresar: numero

    primerNro = numero + 1
    segundoNro = numero + 2
    tercerNro = numero + 3

    mostrar: primerNro
    mostrar: segundoNro
    mostrar: tercerNro
fin
```

En este caso, se ingresa un valor inicial, que se guarda en el elemento "numero". 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 "numero" = 5
2. Luego, se le suma 1 a "numero" (que vale siempre 5) y se lo guarda en una variable llamada "primerNro". Por lo que "primerNro" = 5 + 1 = 6
3. Se le suma 2 a "numero" y se lo guarda en otra variable llamada "segundoNro". Por lo que "segundo_nro" = 5 + 2 = 7
4. Se le suma 3 a "numero" y se lo guarda en una tercera variable. Por lo que "tercerNro" = 5 + 3 = 8
5. Luego se muestran los valores de cada una de las variables:
    a. primerNro = 6
    b. segundoNro = 7
    c. tercerNro = 8

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

1. Se ingresa 99
2. A 99 se le suma 1 (primerNro=100)
3. A 99 se le suma 2 (segundoNro=101)
4. A 99 se le suma 3 (tercerNro=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.

Versión 2:

```
programa MostrarNumerosV2
inicio
    ingresar: numero

    primerNro = numero + 1
    mostrar: primerNro

    segundoNro = numero + 2
    mostrar: segundoNro

    tercerNro = numero + 3
    mostrar: tercerNro
fin
```

Pasemos a analizar la segunda alternativa de solución.

En este caso, la lógica del planteo es el mismo que en la alternativa de Versión 1.

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.

Versión 3:

```
programa MostrarNumerosV3
inicio
    ingresar: numero

    numero = numero + 1
    mostrar: numero

    numero = numero + 1
    mostrar: numero

    numero = numero + 1
    mostrar: numero
fin
```

Para finalizar con el presente ejemplo, pasamos a analizar la tercera versión de la 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 ("primerNro", "segundoNro" y "tercerNro"), sino que todas las operaciones se realizan sobre "numero". Por esto es que no se suman los valores 1, 2 y 3, sino que es el mismo "numero" 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" y perdiendo a media 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 "numero" = 12
2. Luego, se le suma 1 a "numero" que pasa a valer 13
3. Se muestra este valor: 13
4. Se le suma 1 a "numero" que valía 13, por lo que pasa a valer 14
5. Se muestra este valor: 14
6. Se le suma 1 a "numero" 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.

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 media 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.

3. Tips sobre el desarrollo en pseudocódigo

- La indentación implicar correr el inicio de una línea de código hacia derecha (normalmente un "tab" o 4 espacios). Esto también se conoce como tabulado o sangría. TODAS las líneas que se encuentran en un bloque deben estar indentadas. El único bloque que tenemos de momento es "inicio" y "fin", por lo que todas las líneas (instrucciones) que estén adentro, debe empezar con el margen o sangría agregados, como se muestra en los ejemplos y en los Materiales Adicionales de la unidad.
- Hasta el momento, el universo de nuestro pseudocódigo se limita a las siguientes palabras reservadas:
  - programa
  - inicio
  - fin
  - ingresar
  - mostrar
- Además de esto, podemos hacer operaciones, como ya vimos. Si vamos a realizar un cálculo, es probable que lo hagamos para obtener un resultado. Ese resultado lo vamos a guardar en una variable. La asignación siempre se hace colocando la variable que guarda el resultado a la izquierda, luego el símbolo igual ("=") y luego el valor a asignar o la operación. Ejemplo:

```
resultado = 100 + 200

valorA = 1000
valorB = 2000
resultado = valorA + valorB

resultado = valorA + 300

mensaje = "Este es un mensaje o valor alfanumérico por lo que se usa comillas dobles"

nombre = "Ramon Ismael"
```

Las 3 primeras asignaciones manejan valores numéricos. Las otras 2 asignaciones usan valores alfanuméricos (letras, números y otros símbolos), por lo que esos valores deben ir entre comillas dobles ("").

- No tiene sentido hacer operaciones con variables que previamente no hayan recibido un asignación o un ingreso de valores. Ejemplo:

```
programa MostrarValores
inicio
    primero = 100
    segundo = "Hola mundo"
    ingresar: tercero

    //todos OK
    mostrar: primero    //valor numérico asignado
    mostrar: segundo    //valor alfanumérico asignado
    mostrar: tercero    //valor ingresado

    //incorrecto, "cuarto" no tiene valor
    mostrar: cuarto
fin
```

- En las operaciones de asignación nunca se usa el dos puntos (":"), únicamente el igual ("=")
- Como ya vimos, para darle un valor a una variable, podemos optar por 2 caminos:
    - ingresar un valor: dando un valor a una variable usando la palabra reservada "ingresar:" como se ven en los ejemplos, simulando un valor desconocido que sería ingresado por el usuario del programa.
    - asignar un valor: darle un valor fijo a una variable
- No tiene sentido ingresar y luego asignar un valor una variable, porque se perdería el valor ingresado (es "pisado" por la asignación"). Ejemplo:

```
programa PisandoValores
inicio
    //no tiene sentido
    ingresar: dato
    dato = "Antonio Vespucio"
    mostrar: dato     //se muestra Antonio Vespucio

    //no tiene sentido
    dato2 = "Nuñez"
    ingresar: dato2
    mostrar: dato2

    //tiene sentido
    dato3 = "La Máquina"
    mostrar: dato3

    //tiene sentido
    ingresar: dato4
    mostrar: dato4
fin
```

Como regla general, en la mayoría de las veces será un error ingresar y asignar (o viceversa) un valor a una variable en forma sucesiva.

- Cualquier instrucción que no pertenezca a las palabras reservadas antes mencionadas y a las operaciones permitidas, es INVÁLIDO.

  Ejemplo:

```
programa IngresoDeDatos
inicio
    ingresar: nombreCancion
    mostrar: "La canción elegida es "
    mostrar: nombreCancion

    reproducir nombreCancion    //"reproducir" no es una instruccion válida
fin
```

- No se pueden combinar las operaciones de ingreso y asignación. Ejemplo:

```
programa NuevoIngreso
inicio
    ingresar: "nombre"      //una variable JAMAS va entre ""
    ingresar: nombre     //forma correcta

    ingresar: 1200        //esto debería ser una asignación
    numero = 1200        //forma correcta

    mostrar: ingresar: resultado     //no se pueden combinar
    mostrar: "Ingresar resultado"     //forma correcta
    ingresar: resultado

fin
```

- Es posible unificar en una sola línea un mensaje y una varias variables. Esto lo vamos a hacer para "ahorrarnos" líneas de código. Esto lo hacemos usando el símbolo "+", pero va a funcionar como una unión (concatenación) en lugar de una operación aritmética. Ejemplo:

```
mostrar: "Ingresar resultado"
ingresar: resultado
mostrar: "El resultado es " + resultado  //muestra el mensaje
                                         // "El resultado es " y el valor ingresado

apodo = "Mencho"
apellido = "Medina Bello"
mostrar: "El apodo " + apodo + " corresponde al señor " + apellido
                    //muestra el mensaje "El apodo Mencho corresponde al señor Medina Bello"

calculo = 100 + 2000
mostrar: "El valor es " + calculo      //muestra el mensaje
                                        // "El valor es 2100"
```

ACTIVIDAD 2:

Escribir un (1) programa en pseudocódigo que cumpla con el siguiente requerimiento:

- pida el ingreso de un dato
- realice una asignación de una variable
- muestre un mensaje y/o contenido de alguna variable

Condiciones de aceptación adicionales:

- El código debe estar indentado
- Usar únicamente el pseudocódigo visto en el curso
- Que tenga sentido, sea claro, y breve (debe tener EN TOTAL un mínimo 6 líneas y máximo 8 líneas, sin contar espacios en blanco y comentarios)

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

4. Programa resueltos

Requerimientos:

1. Se debe ingresar dos números, sumarlos y mostrar su resultado. Luego ingresar un tercer numero, el cual se sumara al resultado de la suma anterior y mostrarlo el nuevo resultado.

2. Se debe solicitar ingresar 4 números y sacar el promedio de ellos.

3. Se debe solicitar el ingreso de cada día de la semana y se deben mostrar al final los datos ingresados.

4. Se deben mostrar dos números y luego sobrescribirlos con valores nuevos ingresados por el usuario. Mostrar los nuevos números ingresados.

5. Escribir un programa que descontará el stock de productos en un almacén. Se debe mostrar el stock inicial de productos disponibles, descontar la cantidad ingresada por el usuario de productos vendidos, y mostrar finalmente el número de stock actualizado.

Página dejada intencionalmente en blanco

IMPORTANTE: ¿Intentaron resolver los ejercicios antes de ir buscar la solución…?

Si no, deberían intentarlo. Pueden volver a los requerimientos.

Si sí, bien por Ustedes. Prosigan a las resoluciones para verificar el resultado obtenido

Página dejada intencionalmente en blanco

1. Planteo lógico:

```
/*
* - Ingresar un número
* - Ingresar un segundo número
* - Sumar números
* - Mostrar resultado
* - Pedir un nuevo número
* - Sumar la suma anterior con el nuevo número
* - Mostrar el resultado
*/
```

1.1. Algoritmo en pseudocódigo:

```
programa SumaTres
inicio
    mostrar: "Ingrese un número"
    ingresar: nro1
    mostrar: "Ingrese otro número"
    ingresar: nro2
    nroTemp = nro1 + nro2
    mostrar: "Los primero dos números suman:"
    mostrar: nroTemp
    mostrar: "Ingrese el tercer número"
    ingresar: nro3
    nroTemp = nroTemp + nro3
    mostrar: "Los tres números suman:"
    mostrar: nroTemp

fin
```

2. Planteo lógico:

```
/*
* - Ingresar un número, repetir 4 veces.
* - Sumar los cuatro números.
* - Dividir la suma de los numero por la cantidad de números ingresados (4).
* - Mostrar promedio.
*/
```

2.1. Algoritmo en pseudocódigo:

```
Programa PromedioDeCuatro
inicio
    mostrar: "ingrese 1er. número"
    ingresar: nro1
    mostrar: "ingrese 2do. número"
    ingresar: nro2
    mostrar: "ingrese 3er. número"
    ingresar: nro3
    mostrar: "ingrese 4to. número"
    ingresar: nro4
    suma = nro1 + nro2 + nro3 + nro4
    promedio = suma / 4
    mostrar: "El promedio es: " + promedio
fin
```

3. Planteo lógico:

```
/*
* - Ingresar un día
* - Almacenar el dato temporal
* - Repetir para cada día de la semana
* - Mostrar los días de la semana ingresados
*/
```

3.1. Algoritmo en pseudocódigo:

```
programa MostrarDiasDeLaSemana
inicio

    ingresar: nombreDiaUno //se ingresa el primer nombre
    //se almacena el nombre del día
    primerDia = nombreDiaUno
    ingresar: nombreDiaDos //se ingresa el segundo nombre
    //se almacena el nombre del día
    segundoDia = nombreDiaDos
    ingresar: nombreDiaTres //se ingresa el tercer nombre
    //se almacena el nombre del día
    tercerDia = nombreDiaTres
    ingresar: nombreDiaCuatro //se ingresa el cuarto nombre
    //se almacena el nombre del día
    cuartoDia = nombreDiaCuatro
    ingresar: nombreDiaCinco //se ingresa el quinto nombre
    //se almacena el nombre del día
    quintoDia = nombreDiaCinco
    ingresar: nombreDiaSeis //se ingresa el sexto nombre
    //se almacena el nombre del día
    sextoDia = nombreDiaSeis
    ingresar: nombreDiaSiete //se ingresa el séptimo nombre
    //se almacena el nombre del día
    septimoDia = nombreDiaSiete
```

```
    // se muestran los días ingresados
    mostrar: primerDia
    mostrar: segundoDia
    mostrar: tercerDia
    mostrar: cuartoDia
    mostrar: quintoDia
    mostrar: sextoDia
    mostrar: septimoDia

fin
```

4. Planteo lógico:

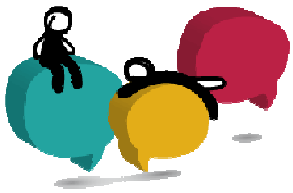
```
/*
* - Asigno dos valores números
* - Los muestro
* - Pido que se ingresen nuevos valores
* - Piso los anteriores y los muestro
*/
```

```
programa CambioDeNumeros

inicio
    primerNumeroAlmacenado = 10
    segundoNumeroAlmacenado = 20

    //mostramos los números almacenados
    mostrar: "El primer número almacenado es"
    mostrar: primerNumeroAlmacenado
    mostrar: "El segundo número almacenado es"
    mostrar: segundoNumeroAlmacenado

    ingresar: primerNumeroNuevo //se ingresa el primer número nuevo
    //guardamos el primer número nuevo
    primerNumeroAlmacenado = primerNumeroNuevo

    ingresar: segundoNumeroNuevo //se ingresa el segundo numero nuevo
    //guardamos el segundo numero nuevo
    segundoNumeroAlmacenado = segundoNumeroNuevo

    //mostramos el valor de los números almacenados
    mostrar: "El primer número almacenado fue cambiado a:"
    mostrar: primerNumeroAlmacenado

    mostrar: "El segundo número almacenado fue cambiado a:"
    mostrar: segundoNumeroAlmacenado
fin
```

5. Planteo lógico:

```
/*
* - Definir stock inicial y mostrarlo
* - Ingresar cantidad de productos vendidos
* - Actualizar stock (inicial - vendidos)
* - Mostrar stock actualizado
*/
```

```
programa ActualizacionDeStock

inicio
    stockActual = 100    //definimos el stock inicial
    //mostramos el stock actual
    mostrar: "El stock actual en el almacén es de:"
    mostrar: stockActual

    mostrar: "Ingresar cantidad de productos vendidos:"
    // se ingresa la cantidad de productos vendidos
    ingresar: cantidadProductosVendidos

    //restamos la cantidad de productos vendidos al stock inicial
    stockActual = stockActual - cantidadProductosVendidos

    //mostramos el stock actualizado
    mostrar: "El stock fue actualizado, el almacén cuenta con un stock de:"
    mostrar: stockActual

fin
```



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 se resuelven, no se suben a los foros)

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

Lo que vimos

- Pseudocódigo
- Convenciones de los mismos
- 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