Tutorial de Pseudocódigo

Esta sección se ha diseñado con un sentido totalmente didáctico, para servir de apoyo a las clases de pseudocódigo o algoritmia que forman parte de los cursos de programación.

Aunque no existen reglas comunes para la escritura de los pseudocódigos, he recogido una notación estándar que se utiliza en la mayor parte de los libros de programación en español.

Las palabras básicas reservadas, es decir, aquellas que pueden ser traducidas a palabras de un lenguaje de programación se presentan en minúscula cursiva.

Si tienes comentarios, críticas o sugerencias, sobre este tutorial, por favor envíame un mensaje.

Datos y Tipos de Datos

Las cosas se definen en la computadora mediante datos, los algoritmos que vamos a diseñar van operar sobre esos datos. A nivel de la máquina los datos se representan como una serie de bits (dígito 1 ó 0). Los tipos de datos que vamos a manejar a lo largo del tutorial son : númericos y carácter (también llamados alfanuméricos), existen además, los datos de tipo lógicos que solo pueden tener uno de dos valores : verdadero o falso.

Los datos numéricos pueden ser de varias clases: enteros, enteros largos, de doble precisión, de simple precisión, de coma flotante, reales; cuya definición depende del lenguaje de programación utilizado.

Los datos de tipo carácter o alfanuméricos están compuestos por el conjunto finito y ordenado de caracteres que la computadora reconoce:

caracteres alfabéticos : A,B,C,.....Z ; a,b,c,.....
caracteres numéricos : 0,1,2, ...9 (que no son números)
caracteres especiales : +, /, *, ?,%,$, #, !, <, >,ä,¦,{},~,etc.

Una cadena o string es una sucesión de caracteres que se encuentran delimitados por comillas (" "). La longitud de la cadena es la cantidad de caracteres que la forma, incluyendo los espacios que son un carácter más. Así:

"Asunción, Paraguay" es una cadena de longitud 18

"Miércoles 7 de Marzo de 2001" es una cadena de longitud 28 (el 7 y el 2001 no son números)

"123456" es una cadena de longitud 6, no es el número 123.456 sobre "123456" no se puede realizar ninguna operación aritmética como sumar, restar, etc, ya que se trata de una cadena alfanumérica.

Variables

Cuando representamos datos, numéricos o alfanuméricos, debemos darles un nombre. Una variable es un nombre que representa el valor de un dato.
En esencia, una variable es una zona o posición de memoria en la computadora donde se almacena información. En un pseudocódigo y también en un programa se pueden crear tantas variables como querramos. Así tenemos:

A = 50; Variable tipo numérica A cuyo valor es 50.
Ciudad = "Asunción"; Variable alfanumérica o de tipo carácter Ciudad, cuyo valor es "Asunción"
X = C + B; Variable numérica X cuyo valor es la suma de los valores de las variables numéricas C y B. Es una variable calculada

Ten en cuenta que las operaciones que se pueden realizar con dos o más variables exigen que éstas sean del mismo tipo. No podemos "sumar", por ejemplo una variable alfanumérica a otra númerica y viceversa como por ejemplo:

```
FechaNueva="1 de Junio de 1.971" + 5
```

Esto no se puede hacer !!

Para dar nombres a las variables hay que seguir ciertas reglas:

Pueden tener hasta 40 caracteres

Debe empezar obligatoriamente con una letra (a-z o A-Z)

No pueden contener espacios en blanco

El resto de los dígitos pueden ser números

Se pueden incluir caracteres especiales como el guión o el punto.

Ejemplos de nombres válidos de variables

FechaNueva
C1
totalGuaranies
CONTADOR-5
H123
cantidad_de_Alumnos
Pedido.Almacen

Ejemplos de nombres de variables NO válidos

Fecha nueva
1contador
24ABC
primer-valor N

Algunos lenguajes de programación exigen la declaración de las variables que se van a utilizar en todo el programa; es decir, que al comenzar el programa se debe decir que nombre tiene, de que tipo es (numerica o alfanumérica) y un valor inicial. Como aqui no estamos tratando con ningún lenguaje, la declaración de las variables puede omitirse.

Las variables también pueden inicializarse; darles un valor inicial. Por defecto, todas las variables para las que no especifiquemos un valor inicial, valen cero si son de tipo numérica y nulo (nulo no es cero ni espacio en blanco; es nulo) si son de tipo carácter.

Operaciones

Las variables se pueden procesar utilizando operaciones apropiadas para su tipo.
Los operadores son de 4 clases:

Relacionales
Aritméticos
Alfanuméricos
Lógicos

Los operadores relacionales se utilizan para formar expresiones que al ser evaluadas producen un valor de tipo lógico: verdadero o falso. Ellos son:

Signo	Operador
>	Mayor que
<	Menor que
=	Igual a
<=	Menor o igual que
>=	Mayor o igual que
<>	Distinto

Ejemplos:

Ejemplo	Resultado
25 <= 25	Verdadero
25 <> 25	Falso
25 <> 4	Verdadero
50 <= 100	Verdadero
500 >= 1	Verdadero
1 = 6	Falso

Cuando se comparan caracteres alfanuméricos se hace uno a uno, comenzando por la izquierda hacia la derecha.
Si las variables son de diferente longitud,pero exactamente iguales, se considera que la de menor longitud es menor.
Los datos alfanuméricos son iguales si y solo si tienen la misma longitud y los mismos componentes.
Las letras minúsculas son mayores que las mayúsculas y cualquier caracter numérico es menor que cualquier letra mayúscula o minúscula; Así:

carácter numérico < mayúsculas < minúsculas.

Ejemplos:

Comparación	Resultado
"A" < "B"	Verdadero
"AAAA" > "AAA"	Verdadero
"B" > "AAAA"	Verdadero
"C" < "c"	Verdadero
"2" < "12"	Falso

Estas comparaciones se realizan utilizando el valor ASCII de cada carácter
Para tratar los números se utilizan los operadores aritméticos:

Signo	Significado
+	Suma
-	Resta
*	Multiplicación
/	División
^	Potenciación
MOD	Resto de la división entera

El único operador alfanumérico se utiliza para unir o concatenar datos de este tipo:

Signo	Significado
+	**Concatenación**

Ejemplos:

Expresión	Resultado
"Pseudo" + "código"	"Pseudocódigo"
"3" + "4567"	"34567"
"Hola " + "que tal ?"	"Hola que tal ?"

Los operadores lógicos combinan sus operandos de acuerdo con las reglas del álgebra de Boole para producir un nuevo valor que se convierte en el valor de la expresión, puede ser verdadero o falso.

Signo	**Significado**
OR	Suma lógica (O)
AND	Producto lógico (Y)
NOT	Negación (NO)

Ejemplos:

Expresión	**Resultado**
Verdad AND Falso	Falso
NOT Falso	Verdad
Verdad OR Falso	Verdad

Por ejemplo, la expresión: (12 + 5) OR (7 + 3) = 10 es verdadera (se cumple una y verdad OR Falso es Verdad).
La expresión (12 * 5) AND (3 + 2) = 60 es falsa
(verdad AND falso = Falso).

¿Cómo se evalúan los operadores? La prioridad de los operadores es:

1. Paréntesis
2. Potencias
3. Productos y Divisiones
4. Sumas y restas
5. Concatenación
6. Relacionales
7. Lógicos

ASIGNACIONES

La operación de dar valor a una variable se llama asignación. La asignación vamos a representarla con el símbolo <-- ; una flecha apuntando a la izquierda. No utilizaremos en signo = ya que el operador de asignación varía de acuerdo con el lenguaje de programación utilizado. El formato general de una asignación es:

nombre de la variable ✍ ***expresión***

La flecha se sustituye en los lenguajes de programación por = (basic); : = (pascal). Pero para evitar ambigüedades en el pseudocódigo utilizaremos la flecha para la asignación y el símbolo = ***par***a indicar igualdad. He aquí algunos ejemplos:

A ✍↴ 100 ; significa que a la variable A se le ha asignado el valor 100, ahora A vale 100.
suma ✍ 5+10; asigna el valor 15 a la variable suma a través de una asignación aritmética.
x ✍ z + v ; asigna el valor de la suma de las variables z y v a la variable x. El resultado depende de los valores que se asignen a x y a z

Toda asignación es destructiva. Esto quiere decir que el valor que tuviera antes la variable se pierde y se reemplaza por el nuevo valor que asignamos, así cuando se ejecuta esta secuencia:

B ✍ 25
B ✍ 100
B ✍↴ 77

el valor final que toma B será 77 pues los valores 25 y 100 han sido destruidos.

Cuando una variable aparece a ambos lados del símbolo de asignación como:

C ✍ C + 1

conviene incializarlas al comenzar el programa con cero, aunque no es obligatorio por ahora (en algunos lenguajes de programación sí es necesario).

Recordemos que no se pueden asignar valores a una variable de un tipo diferente al suyo.

Pongamos atención a este ejemplo de asignaciones:

A ✍ A + 2 * B
B ✍ C - A

En las dos primeras acciones, A toma el valor 3 y B el valor 4.
C ✍ A + 2 * B

La expresión tomará el valor 3 + 2 * 4 = 3 + 8 = 11
C vale entoces 11.
B ✍ C - A

C vale 11, A vale 3, por lo tanto B valdrá 11 - 3 = 8
Como toda asignación es destructiva, el valor anterior de B se pierde y pasa a valer ahora 8.

Otro ejemplo:

J ✍ J * 3

Que valor tiene J al final ? Veamos.
Primero se asigna 33 a la variable J, J vale entonces 33; luego:

J ✍ J + 5

Esto es: Sumar 5 al valor de J y asignarlo a la variable J. J vale 33.
J↴ 33 + 5 ; J ↴ 38
J vale ahora 38.
El valor anterior que era 33 se destruyó.
Seguimos: **J ✍ J * 3**
Esto es: Multiplicar 3 al valor de J y asignarlo a la variable J.
J ✍ 38 * 3 ; J ✍ 114
El valor final de J es 114.

Entrada y Salida de Información

Los datos que vamos a obtener del usuario para procesarlos también deben ser asignados a variables, la operación de lectura, lee un dato y lo asigna a una variable. La instrucción para la lectura de un dato es *leer* o también *ingresar*. Por ejemplo:
***leer* numero**

Esta instrucción pide al usuario un valor que será asignado a la variable numero, es decir, en numero se almacena el valor ingresado por el usuario
***leer* Edad, Peso, Sexo**

Representa la lectura de tres valores que se van a almacenar en las variables Edad, Peso y Sexo; en ese mismo orden.

Ya tenemos nuestro primer comando: *leer*

Ahora bien, cuando queramos mostrar el resultado del algoritmo, un mensaje, un valor, etc, vamos a utilizar el comando *imprimir*. Por ejemplo:
imprimir "Hola" ; muestra en la pantalla el mensaje Hola, Hola va entre comillas porque es una cadena.
imprimir A; muestra en la pantalla el valor que está almacenado en la variable A.
imprimir "El valor del promedio es:", promedio
Esta instrucción muestra el mensaje que está entre comillas y luego el valor de la variable

promedio. La coma separa el mensaje de la variable. Si promedio vale 5, lo que se verá en la pantalla será:

El valor del promedio es: 5

Ya conocemos dos comandos que vamos a utilizar en nuestros pseudocódigos: *leer* e *imprimir*

También podemos mostrar un mensaje cuando solicitamos algún dato al usuario por medio del comando leer así: *leer* "Ingrese su edad", edad

El valor de la variable que pedimos al usuario se asigna a edad. Esta instrucción se verá así en la pantalla: Ingrese su edad ?

El símbolo de interrogación aparece automáticamente cada vez que usamos el comando *leer*.

Entonces, en la escritura de pseudocódigos, las acciones de lectura y escritura se representan por los siguientes formatos:

***leer* Variable o lista de variables separadas por comas.** Ejemplos:

leer Edad
leer Ciudad, Pais

***imprimir* Variable o lista de variables separadas por comas.**

Ejemplos:

imprimir promedio
imprimir TotalMes, TotalAño, TotalGeneral

imprimir "Así se muestra un mensaje o comentario"

En resumen: Las instrucciones disponibles para escribir un programa dependen del lenguaje de programación utilizado. Existen instrucciones -o acciones- básicas que se pueden implementar de modo general en cualquier algoritmo y que soportan todos los lenguajes de programación. Estas son:

1- Instrucciones de inicio/fin
2- Instrucciones de asignación
3- Instrucciones de lectura
4- Instrucciones de escritura

Tipo de Instrucción	**Pseudocódigo**
Comienzo de proceso	*inicio*
Fin de proceso	*fin*
Entrada (Lectura)	*leer*
Salida (Escritura)	*imprimir o escribir*
Asignación	✍

Resolución de Problemas

Antes de resolver un problema por medio de un pseudocódigo, es necesario definirlo y comprenderlo claramente. Leeremos con atención el enunciado del problema y una vez comprendido responderemos a las preguntas:

¿ Qué información debe proporcionar la resolución del problema?

¿ Cuáles son los datos que necesito para resolver el problema?

La respuesta de la primera pregunta nos dice que salidas va a proporcionar el algoritmo y la segunda qué datos se nos proporcionan para resolver el problema y cuáles debemos calcularlos.

Problema: **Leer las longitudes de un rectángulo y calcular la superficie y el perímetro.**

Para calcular el área y el perímetro de un rectángulo, se necesitan las medidas del ancho y el alto, estas medidas serán leídas en dos variables.
Las salidas serán los valores del área y el perímetro que serán calculados utilizando fórmulas.

Entradas: largo, ancho

Salidas: perímetro, área
El pseudocódigo es:

```
inicio
        leer largo
        leer ancho
        perimetro ✍ largo + ancho * 2
        area ✍ largo * ancho
        imprimir perimetro
        imprimir area
fin
```

Problema: **Escribir un pseudocódigo que intercambie el valor de dos variables.**

Si se tienen, por ejemplo A = 5 y B = 10, se quiere intercambiar el valor de las variables, así:
A = 10; B = 5.

No podemos asignar directamente el valor de una a la otra porque uno de los valores se destruiría; de modo que esto no se puede hacer:

A ✍ B (el valor de A se pierde y quedaría A = 10 ; B = 10)

La solución consiste en asignar el valor de una de las variables a otra variable auxiliar.

```
inicio
leer A, ✍ A
A ✍ B
B ✍ Auxiliar
imprimir A, B
fin
```

Sigamos paso a paso el pseudocódigo:

leer A, B ..... Se pide al usuario dos valores. Supongamos que se ha ingresado A = 100 ; B = 5
Auxiliar ✍ A ..... Se asigna a Auxiliar el valor 100. Auxiliar vale 100.
El valor de las variables es:

A
B
Auxiliar
100
5
100

A ✍ B ..... Se asigna a A el valor de B para intercambiar. Ahora el valor de las variables es:

A
B
Auxiliar
5
5
100

B ✍ Auxiliar ..... El valor de A que se guardó en Auxiliar se asigna a B para el intercambio.

A
B
Auxiliar
5
100
100

El intercambio está hecho. Luego se imprimen los respectivos valores ya intercambiados con la línea:
imprimir A, B

Contador

Un contador es una variable destinada a contener valores que se van incrementando o decrementando cada vez que se ejecuta la acción que lo contiene. El incremento o decremento es llamado paso de contador y es siempre constante.
Por ejemplo; el marcador de un partido de fútbol, cada vez que un equipo anota un gol, aumenta su marcador en una unidad.
En las carrera de automóviles, cada vez que un vehículo pasa por la línea de meta, se incrementa en una unidad el número de vueltas dadas al circuito, o bien se decrementa en una unidad el número de vueltas que quedan por realizar.

Aunque el incremento es siempre constante, el paso de contador no necesariamente puede ser la unidad como en los ejemplos que se han dado más arriba; también puede incrementarse o decrementarse a de dos, tres, cuatro,.... n; es decir, puede ser cualquier número que conserva el mismo valor durante todo el programa.
La sintaxis de una variable contador es: **C ✍ C + 1 o C ✍ C - 1**

variable ✍ variable + constante (al incrementar)
variable ✍ variable - constante (al decrementar)

Ejemplos:

gol_local ✍ gol_local + 1

vueltas ✍ vueltas + 1

faltan ✍ faltan - 1

de_cinco ✍ de_cinco + 5

c ✍ c + 1

x ✍ x - 3

Observación: Cuando una variable aparece a ambos lados del símbolo de asignación, conviene inicializarlas a cero.

Acumulador o Sumador

Es una variable que nos permite guardar un valor que se incrementa o decrementa en forma NO constante durante el proceso. En un momento determinado tendrá un valor y al siguiente tendrá otro valor igual o distinto. Por ejemplo; cuando realizamos un depósito en el banco, la cantidad depositada cada vez no es siempre la misma; unas veces será una cantidad y otras veces distinta. Lo mismo ocurre cuando realizamos algún retiro, pero decrementando la cantidad total.

La sintaxis es:

acumulador ✍ acumulador + variable (al incrementar)
acumulador ✍ acumulador - variable (al decrementar)

acumulador es la variable en la que se almacena el resultado.
variable contiene el número que estamos incrementando o decrementando

Ejemplos:

saldo ✍ saldo + entrega
saldo ✍ saldo - retiro
suma ✍ suma + numero
A ✍ A + edad

ESTRUCTURA DE SELECCIÓN

La estructura de selección, se ejecuta condicionalmente, es decir, si una cierta condición es verdadera se ejecuta un bloque de instrucciones, si es falsa se ejecuta un bloque diferente de instrucciones. Por ejemplo, si en el cine proyectan "Star Wars Episode I", entonces hay que formar fila para comprar los billetes e ingresar al cine, si no, decidimos otra actividad como ir a bailar.

Si utilizamos una selección es para indicar que según el resultado cierto o falso de una expresión vamos a tomar una decisión de realizar determinadas acciones especificadas; seleccionamos las acciones a realizar.
La instrucción que permite tomar una decisión, basada en una condición es **Si...Entonces**. Al evaluar la condición, **Si...entonces** puede devolver solo dos resultados posibles: Verdadero o Falso; es decir, Si o No. El formato de la estructura de selección es:
si* <condición> *entonces

instrucción 1
instrucción 2
.....
instrucción n

si-no

instrucción a
instrucción b
.....
instrucción z

fin-si

Observa como el sangrado permite identificar fácilmente que grupo de instrucciones se ejecutan en cada caso.

Por ejemplo, Cuando realizamos una llamada telefónica:

Si* {señal de ocupado} *entonces
Colgar el teléfono
si - no
Iniciar la conversación
fin - si

En este caso, la condición es {señal de ocupado}, que puede ser verdadera o falsa. Si es verdadera, entonces debemos colgar el teléfono y si no, podemos relizar la conversación.
Ejemplo:

Si A = 5 entonces
imprimir"A es 5"
si - no
imprimir "A no es igual a 5"
fin - si

Tambien puede obviarse el si - no cuando no nos interesa ejecutar un bloque de instrucciones en caso de que la condición no se cumpla.

Si {condición} entonces
instrucción 1
instrucción 2
.....
instrucción n
fin - si

Por ejemplo;
Si {hoy es Miércoles} entonces
Comprar entradas para el cine
fin - si

Ejemplos

Introducir un número por teclado y determinar si es positivo o negativo.-
Para saber si un núero es positivo o negativo, debemos saber si es menor o mayor a cero. Si es mayor, el número es positivo y si es menor resulta negativo. Utilizamos ***Si…*** *para evaluar como es el número con respecto a cero y mostramos los mensajes correspondientes en cada caso. Así:*

```
inicio
leer Numero
Si Numero < 0 entonces
  imprimir "El número es negativo"
si-no
  imprimir "El número es positivo"
fin-si
fin
```

Ejemplo 2. ***Dados dos números, establecer cuál es mayor .***
Comenzamos leyendo ambos números, que en el ejemplo se llamarán NumeroA y NumeroB. Luego debemos comparar como es uno contra el otro (puede ser NumeroA contra NumeroB o bien comparar NumeroB contra NumeroA):

```
inicio
leer NumeroA, NumeroB
Si NumeroA < NumeroB entonces
imprimir "El mayor es:", NumeroB
si-no
imprimir "El mayor es:", NumeroA
fin-si
fin
```

En este ejemplo, que pasaría si los números fueran iguales?. Hagamos la prueba
Luego de leer los números, por ejmplo: NumeroA=100 y NumeroB=100 se ejecutan las instruciones:
Si NumeroA < NumeroB entonces
imprimir "El mayor es:", NumeroB
El resultado de la condición
Por lo tanto, al ser falsa la condición, se ejecuta la instrucción *imprimir "El mayor es:", NumeroA.*
Por tanto, el algoritmo ofrecerá una solución incorrecta cuando los números son iguales. Para solucionar esto, tenemos que prever el caso de que los números sean inguales.

```
inicio
leer NumeroA, NumeroB
Si NumeroA < NumeroB entonces
    imprimir "El mayor es:", NumeroB
si-no
      Si NumeroA > NumeroB  entonces
           imprimir "El mayor es:", NumeroA
      si-no
          imprimir "Los números son iguales"
      fin-si
fin-si
fin
```

Esta solución contiene dos estructuras de repetición, una dentro de la otra (anidada). En caso de ser necesario podemos anidar tantas estructuras de selección como sea necesario. El algoritmo averigua si A es menor a B, si no lo es, tenemos otras dos posibilidades: que sea menor o igual, esto es lo que determina la estructura anidada.
Otro ejemplo de estructuras de reperición anidadas, consiste en dado un número del 1 al 7, establecer al día de la semana.

```
inicio
   leer numero
```

```
   Si numero=1 entonces
      imprimir "Domingo"
   si-no
      Si numero=2 entonces
         imprimir="Lunes"
      si-no
         Si numero=3
            imprimir "Martes"
         si-no
            Si numero=4 entonces
               imprimir "Miércoles"
            si-no
               Si Numero=5 entonces
                     imprimir "Jueves"
               si-no
                     Si numero=6 entonces
                        imprimir "Viernes"
                     si-no
                        Si numero=7 entonces
                           imprimir "Sábado"
                        si-no
                           imprimir "El número debe estar entre 1 y 7"
                        fin-si
                     fin-si
                fin-si
               fin-si
            fin-si
         fin-si
      fin-si
 fin
```

Notarás que tenemos varios *Si...entonces* anidados, ya que si el número ingreso no es 1, tenemos que preguntar si es 2 ó 3 ó 4...etc. El último *Si...entonces* es para verificar que el número ingresado no es 1, 2, 3, 4, 5, 6 ó 7; sino cualquier otro que no nos interesa.
Repasa los algoritmos anteriores.
Resulta bastante tedioso anidar un montón de Si ... entonces, como en el ejemplo del día de la semana. Cuando queramos o necesitemos hacer numerosas comparaciones podemos usar otra estructura de selección llamada *Según Sea.* El formato de estructura de selección *Según sea* es:

***Según sea* <variable>**
   ***Caso* = <condición>**
   ***Caso* = <condición>**
         instrucción o instrucciones
   Otro caso
         instrucción o instrucciones
fin-según

Así, utilizando esta estructura, el problema del día de la semana será así:

```
inicio
   Leer numero
   Según sea numero
      Caso = 1
                  imprimir "Domingo"
      Caso = 2
         imprimir="Lunes"
Caso = 3
         imprimir="Martes"
Caso = 4
         imprimir "Miércoles"
Caso = 5
         imprimir "Jueves"
Caso = 6
         imprimir "Viernes"
```

```
Caso = 7
        imprimir "Sábado"
Otro Caso
        imprimir "El número debe estar entre 1 y 7"
   fin-según
fin
```

Lo cual resulta menos engorroso que varios *Si... entonces* anidados. Es posible anidar *Si... entonces* dentro de estructuras *Según sea* y viceversa.
Observa que la instrucción *Otro Caso* ejecuta una o varias instrucciones cuando no se cumple ningún caso de los contemplados más arriba. *Otro Caso* debe estar siempre al final (cuando sea necesario, si no o es se puede omitir *Otro Caso*)
El uso de una u otra estructura depende de cada persona, pero en general cuando las condiciones a evaluar son muchas, se utiliza *Según Sea*.-

La estructura según sea admite varias condiciones por ejemplo:

```
Según sea MES
      caso= 1,3,5,7,8,10,12
            TDias ✍ 31
      caso = 2,4,6,11
            TDias ✍ 30
      caso = 2
            TDias ✍ 28
fin-según
```

Este pequeño ejemplo establece el número de días de un mes determinado almacenado en la variable MES (para años no bisiestos). En lugar de escribir varios *Caso= 1, Caso =2*, etc, se puede especificar acción o acciones (en este caso la asignación de dias a *TDias*) cuando la variable tome uno de los valores separados por comas. Es decir si *TDias* es 1 ó 3 ó 5 ó 7 ó 8 ó 10 ó 12; se ejecuta *TDias*=31.

REPETICION

La estructura repetitiva se utiliza cuando se quiere que un conjunto de instrucciones se ejecuten un cierto número finito de veces. Llamamos bucle o ciclo a todo proceso que se repite un cierto número de veces dentro de un pseudocódigo o un programa.

Existen dos tipos de estructuras repetitivas; la primera es aquella en donde se tiene perfectamente establecido el número de veces que un grupo de acciones se van a ejecutar (20, 5, 2 veces), y la segunda en la que el número de repeticiones es desconocido y se hará hasta que se cumpla o no cierta condición.

Un ejemplo de la primera sería imprimir los datos de los alumnos de una clase (se conoce cuantos alumnos hay) y un ejemplo de la segunda puede ser el mostrar un mensaje de error cada vez que el usuario pulse una determinada tecla (no sabemos cuantas veces pulsará esa tecla).
Las acciones que forman parte del cuerpo del bucle son ejecutadas de forma repetitiva mediante la ocurrencia o no de una condición.

Cuando conocemos de antemano el número de veces en que se desea ejecutar una acción o grupo de acciones, se utiliza la estructura repetitiva Desde o Para.
Esta estructura ejecuta las acciones del cuerpo del bucle un número especificado de veces, y de modo automático controla el número de iteraciones o pasos.

La sintaxis es:

***Desde* variable ✍ Vi *hasta* Vf [incremento]**
acción o acciones

fin-desde

Donde:

variable: variable índice
Vi: valor inicial de la variable índice
Vf: valor final de la variable índice
[incremento]: el número que se incrementa (o decrementa) a la variable índice en cada iteración del bucle, si se omite es 1.

Ejemplo:
Imprimir todos los números del 1 al 100.

```
Inicio
      desde I ✍ 1 hasta 100
            imprimir I
      fin-desde
fin
```

I es la variable índice con un valor inicial de 1, se incrementa uno en cada paso hasta 100.
Podemos notar que la estructura desde comienza con un valor inicial de la variable índice y las acciones se ejecutan hasta que el valor inicial sea MAYOR que el que el Valor final.
La variable índice se incremente en uno (en el ejemplo) y si este nuevo valor del índice no es mayor que el valor final, se ejecuta de nuevo la acción imprimir.
En este caso se visualizará los números 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ....97, 98, 99, 100 El incremento o paso de contador es siempre 1 si no es especifica lo contrario.

Otro Ejemplo: *Imprimir todos los números pares desde 2 hasta el 300*

```
Desde I ✍ 2 hasta 300 incremento 2
  imprimir I
fin-desde
```

Donde:

La variable índice comienza en 2, se imprime 2 y en el siguiente paso se incrementa (suma) 2 a la variable índice que pasa a valer 4; se imprime el 4 y como 4 es menor que 300 (valor final) , se pasa a una nueva iteración incrementando nuevamente en 2 el índice, que ahora vale 6; y así sucesivamente...
Aquí se visualizan: 2, 4, 6, 8, 10, 12, 14, 16, 18, ... 296, 298, 300

Si deseamos mostrar los *impares* el algoritmo es el siguiente:

```
Desde I ✍ 1 hasta 300 incremento 2
imprimir I
fin-desde
```

La variable índice toma estos valores:

Indice o Paso (veces que se ejecuta el ciclo)	Valor de I
1	1
2	3
3	5
4	7
5	9
6	11
….	….
150	299

Vemos los valores: 1, 3, 5, 7, 9, 11, 13, ... , 299

El ciclo termina mostrando 299 puesto que en el siguiente paso, La variable I valdría 301 y es mayor al límite establecido de 300. I pasa a valer realmente 301 en el último paso, solo que la instrucción de imprimir no se ejecuta porque el limite de 300 se supera.
Si diéramos la instrucción de imprimir el valor de I, inmediatamente después del fin-desde, veíamos 301.

Ejemplo 3: *Imprimir los valores comprendidos entre el 460 y 500 en orden inverso.*

Debemos mostrar: 500, 499, 498, 497, ... 462, 461, 460. En este caso haremos un decremento a la variable índice (no un incremento como en los ejemplos anteriores). Tenemos que comenzar nuestra variable índice en 500 y decrementar una unidad hasta alcanzar el 460, así:

***Desde* I ✍ 500 *hasta* 460 *incremento* -1**
***imprimir* I**
fin-desde

Indice o Paso (veces que se ejecuta el ciclo)	Valor de I
1	500
2	499
3	498
4	497
5	496
....	....
39	462
40	461
41	460

Como salida tenemos, entonces: 500, 499, 498, 497, 496, 495, 494, ... 464, 463, 462, 461, 460.

El segundo tipo de estructura repetitiva se diferencia de la primera en que no se conoce el número de repeticiones o iteraciones en que se va a ejecutar una instrucción o un bloque de instrucciones.
Estas estructuras son básicamente dos: Estructura **mientras....fin-mientras** y la estructura **repetir.... hasta**. Estas dos se diferencian en que la verificación de la condición para repetir el ciclo se hace al inicio con *mientras* y al final con *repetir*
También existen estructuras repetitivas que son combinaciones de estas dos que mencionamos, pero aquí no las estudiaremos.

Estructura Mientras

Como su nombre lo indica, esta estructura repite el cuerpo del bucle **mientras** se cumpla una determinada condición. Su sintaxis es:

***mientras* {condición}**
acción 1
acción 2
acción 3
.....
acción n

fin mientras
instrucción X

Lo primero que el computador hace es examinar la condición, lo que puede dar como resultado dos posibilidades:

- La condición se cumple: Se ejecutan acción 1, acción 2, acción 3, ..., acción n.

Las estará repitiendo hasta que la condición no se cumpla, entonces se sale del ciclo y se siguen ejecutando la o las instrucciones que vienen a continuación y están fuera del bucle; instrucción X.

- La condición no se cumple: No entrará en el ciclo. Se ejecutan las instrucciones que vienen después del bucle, instrucción X, por ejemplo.

De esto se deduce que el cuerpo del bucle de una estructura mientras puede repetirse **cero o más veces**, veces que son determinadas por el cumplimiento o no de la condición.

Ejemplo

mientras **contraseña < > "josua"**
    imprimir **"La contraseña es incorrecta !"**
fin-mientras

imprimir **"Ha ingresado la contraseña correcta"**

Veremos más ejemplos de esta estructura en la sección ejercicios. Al analizarlos coprenderemos mejor como funciona.-

Estructura Repetir

La estructura repetir cumple la misma función que la estructura mientras. La diferencia está en que la estructura mientras comprueba la condición al inicio y repetir lo hace al final; por eso la estructura repetir se ejecuta **por lo menos una vez.**.
La sintaxis es:

repetir
intrucción 1
instrucción 2
instrucción 3
.....
hasta **{condición}**
instrución X

Repetir es opuesta a la estructura mientras. Repetir se ejecuta hasta que se cumpla una condición que se comprueba al final del bucle. Esto implica que las instrucciones que forman el cuerpo del bucle se ejecutan por lo menos una vez. Con la estructura mientras el bucle puede ejecutarse 0 o más veces.

Lo que la computadora hace al ejecutar la estructura repetir es:

- Se ejecutan: instrucción 1, instrucción 2, instrucción 3, ...
- Se evalúa la condición. Si esta es **FALSA** se vuelve a repetir el ciclo y se ejecutan instrucción 1, instrucción 2, instrucción 3, ...
Si la condición es **VERDADERA** se sale del ciclo y se ejecuta instrucción X.

Recordemos una vez más las diferencias entre las estructuras mientras y repetir

MIENTRAS	REPETIR
Comprobación de la condición al inicio, antes de entrar al bucle	Comprobación de la condición al final, después de haber ingresado una vez al bucle
Las instrucciones del cuerpo del bucle se ejecutan en forma repetitiva si la condición es verdadera	Las instrucciones del cuerpo del bucle se ejecutan si la condición es falsa
Las acciones del bucle se pueden ejecutar 0 o más veces	Las acciones del bucle se ejecutan por lo menos una vez

Ejemplo

repetir
imprimir **"La contraseña es incorrecta !"**

***hasta* contraseña = "josua"**

Más ejemplos en la sección Ejercicios.

En resumen, hemos visto dos tipos de estructuras repetitivas, el primer tipo en la que conocemos el número de veces que se repetirá el bucle o ciclo (***Desde ....fin-desde***; y el segundo tipo en el cual no conocemos el numero de veces en se repite el ciclo ya que está determinado por el cumplimiento o no de una condición (***mientras ..... fin-mientras*** y ***repetir....hasta***).

Toda estructura ***Desde....fin-desde*** tiene una estructura ***mientras....fin-mientras*** o ***repetir.....hasta*** equivalente. Sin embargo no toda estructura ***mientras...*** o ***repetir ...*** tiene un ***Desde ....fin-desde*** equivalente.

VECTORES

Hasta ahora hemos trabajado con datos simples que representaban un número, un carácter o una cadena. Sin embargo, en ocasiones se necesita procesar una colección de valores que están relacionados entre sí por algún método, por ejemplo, una lista de calificaciones, de los meses del año, temperaturas a lo largo de una semana, etc.
El procesamiento de estos datos utilizando datos simples es muy difícil. Por eso, se han definido en la programación varias estructuras de datos, que son una colección caracterizada por alguna organización y por las operaciones que se definen en ella.
Una de estas estructuras son los vectores.
Un vector es un conjunto de elementos del mismo tipo que comparten un nombre común; algo así como una variable que puede almacenar al mismo tiempo más de un valor.
Los vectores reciben también el nombre de tablas, listas o *arrays*.
Un vector es un conjunto ordenado y homogéneo. Ordenado porque el primer elemento, segundo, tercero... n-ésimo puede ser identificado y homogéneo porque sus elementos son todos del mismo tipo (numéricos o alfanuméricos, pero no una combinación de ambos).
Gráficamente, un vector se representa como una tabla:

De igual forma que cualquier variable, un vector debe tener un nombre.

Aqui hemos llamado A a nuestro vector ejemplo.
Los elementos que están en el vector A ocupan todos una determinada posición dentro de él:

Así, el número -5 se encuentra en la posición 3; el 99 en la posición 10 y el 12 en la posición 1.
A(3) = -5
A(10) = 99
A(1) = 12

Vemos, entonces que un elemento se referencia por el nombre del vector y la posición que ocupa dentro de él. El número que se coloca entre paréntesis se llama índice y designa la posición del elemento en el vector.
Cada elemento del vector se puede procesar como si fuera una variable simple.
La dimensión de un vector está dada por la cantidad de elementos que contiene y debe ser definida al comenzar el programa.

Cargar un vector