METODOLOGÍA DE LA PROGRAMACIÓN

MÓDULO: Programación en Java

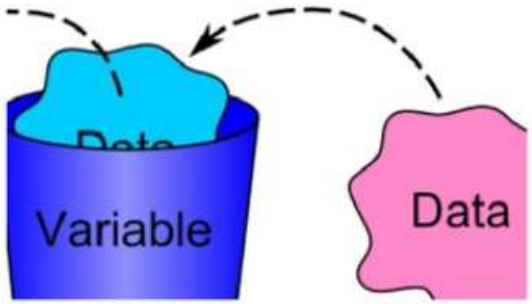
ÍNDICE DE CONTENIDOS

1. VARIABLES

Las variables son secciones reservadas en memoria para almacenar datos los cuales pueden cambiar durante la ejecución del programa. Las variables no son de un tipo único, sino que existen varios tipos de datos informáticos, en función de los cuales queda determinado el valor que puede tomar una variables, así como las operaciones que se pueden realizar sobre ellas. Los tipos de datos más comunes son: enteros, carácter, cadenas de caracteres, booleanos ...

Creamos variables las cuales son identificadas mediante una etiqueta, asignándoles:

- Un nombre.
- Un valor inicial.
- Modificar el valor durante la ejecución del programa.



1. VARIABLES

Es importante, conocer la diferencia que existe entre variables y constantes, puesto el uso y aplicación de estas es diferentes:

• **Constantes**. Se trata de elementos que no varían durante la ejecución del programa. Existe la posibilidad de asignarles un nombre, generalmente al principio del algoritmo, de forma que su denominación y su localización a lo largo de la ejecución sea más cómoda. Dependiendo del tipo de datos al que le asignemos un nombre, podemos encontrarnos con constantes numéricas, tanto enteras como reales, alfabéticas o alfanuméricas. Ejemplo: pi =3.1416.

• **Variable**. Es un espacio en la memoria de la computadora que permite almacenar temporalmente un dato durante la ejecución de un proceso, su contenido puede cambiar durante la ejecución del programa. Para poder reconocer una variable en la memoria de la computadora, es necesario darle un nombre con el cual podamos identificarla dentro de un algoritmo.

2. TIPOS DE DATOS

- **Booleanos.** El tipo de dato lógico o booleano es aquel que representa valores de lógica binaria, esto es 2 valores, que normalmente representan falso o verdadero. Estos se pueden combinar en expresiones lógicas mediante los operadores lógicos (and, or, not).

- **Enteros.** Este tipo de datos se utilizan para representar un subconjunto finito de los números enteros. Los tipos de dato entero disponibles y su tamaño dependen del lenguaje de programación usado así como la arquitectura en cuestión.

- **Carácter y cadenas de caracteres.** En terminología informática y de telecomunicaciones, un carácteres un símbolo que representa cada carácter de un lenguaje natural. Un ejemplo de carácter es una letra, un número o un signo de puntuación.

En programación, una cadena de caracteres o frase (string en inglés) es una secuencia ordenada de longitud arbitraria de elementos que pertenecen a un cierto alfabeto. En general, una cadena de caracteres es una sucesión de caracteres (letras, números u otros signos o símbolos).

En programación, las cadenas pueden estar formadas por cualquier combinación finita de todos los caracteres disponibles (las letras de la 'a' a la 'z' y de la 'A' a la 'Z', los números del '0' al '9', el espacio en blanco ' ', símbolos diversos '!', '@', '%', entre otros).

3. OPERADORES

Un operador es una herramienta que nos ayudara a relacionar uno o dos valores (datos) ante una expresión que nosotros indiquemos y evaluará el resultado entre ambos valores. Por ejemplo, si queremos saber si un dato es igual a otro, utilizaremos un operador para averiguarlo, y este nos regresará una valor de falso o verdadero. Si queremos sumar dos valores, utilizaremos un operador para realizar dicha operación, y nos regresará el valor de la suma. Como vemos, el operador hará interactuar al dato o los datos ante una expresión, y nos indicara el resultado de la operación.

Podemos utilizar los siguientes:

- Aritméticos.
- De comparación.
- De asignación.
- Lógicos.

3. OPERADORES

3.2. OPERADORES DE COMPARACIÓN

Estos operadores se utilizan para comparar. Compararemos valores de variables por ejemplo para saber si una variable contiene el mismo valor que otra variable. Estos operadores se utilizan para comprobar si una condición se cumple o no.

.

Operadores	Uso de los símbolos	Significado
== (igual que)	a == b	Se cumple si 'a' es igual que 'b'
!= (distinto a)	a != b	Se cumple si 'a' es distinto que 'b'
< (menor que)	a < b	Se cumple si 'a' es menor que 'b'
> (mayor que)	a > b	Se cumple si 'a' es mayor que 'b'
<= (menor o igual que)	a <= b	Se cumple si 'a' es menor o igual que 'b'
>= (mayor o igual que)	a >= b	Se cumple si 'a' es mayor o igual que 'b'

3. OPERADORES

3.3. OPERADORES DE ASIGNACIÓN

Estos operadores nos permiten, asignar valores a una variable dada.

3. OPERADORES

3.4. OPERADORES LÓGICOS

- El operador "&&" equivale al "AND" o "Y"; devuelve **true** sólo si los dos operandos **true** o lo que es equivalente, distintas de cero. En cualquier otro caso el resultado es **false**.

- El operador "||" equivale al "OR" u "O inclusivo"; devuelve **true** si cualquiera de las expresiones evaluadas es **true**, o distinta de cero, en caso contrario devuelve **false**.

- El operador "!" es equivalente al "NOT", o "NO", y devuelve **true** cuando la expresión evaluada es **false** o cero, en caso contrario devuelve **false**.

3. OPERADORES

3.4. OPERADORES LÓGICOS

El operador "&&" equivale al "AND" o "Y"; devuelve **true** sólo si los dos operandos **true** o lo que es equivalente, distintas de cero. En cualquier otro caso el resultado es **false**.

El operador "||" equivale al "OR" u "O inclusivo"; devuelve **true** si cualquiera de las expresiones evaluadas es **true**, o distinta de cero, en caso contrario devuelve **false**.

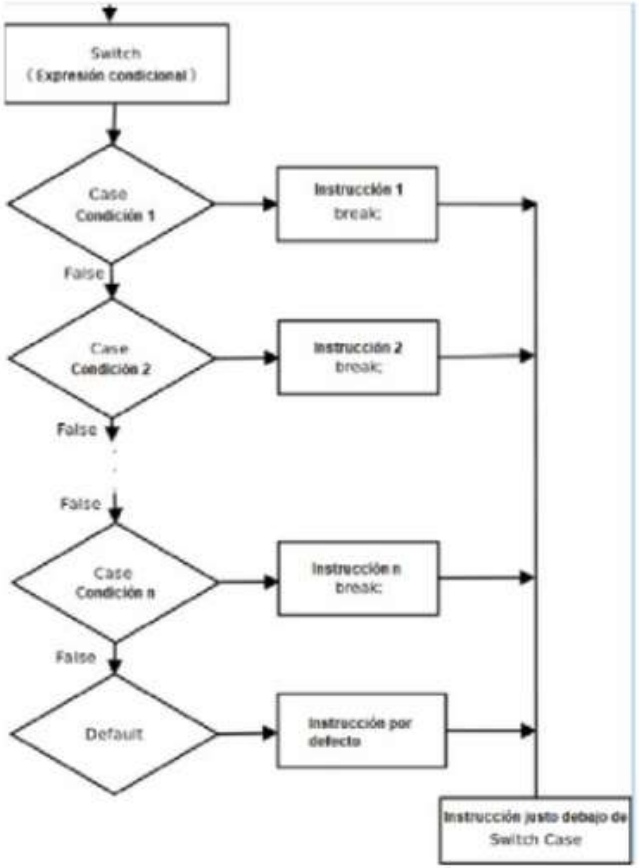
El operador "!" es equivalente al "NOT", o "NO", y devuelve **true** cuando la expresión evaluada es **false** o cero, en caso contrario devuelve **false**.

Operador	Significado
Operador &&	Operador lógico and
Operador ||	Operador lógico or
Operador !	Operador lógico not

4. CASE

Es una estructura que permite analizar los posibles valores que puede tomar una variable, cuando se pueden intuir.

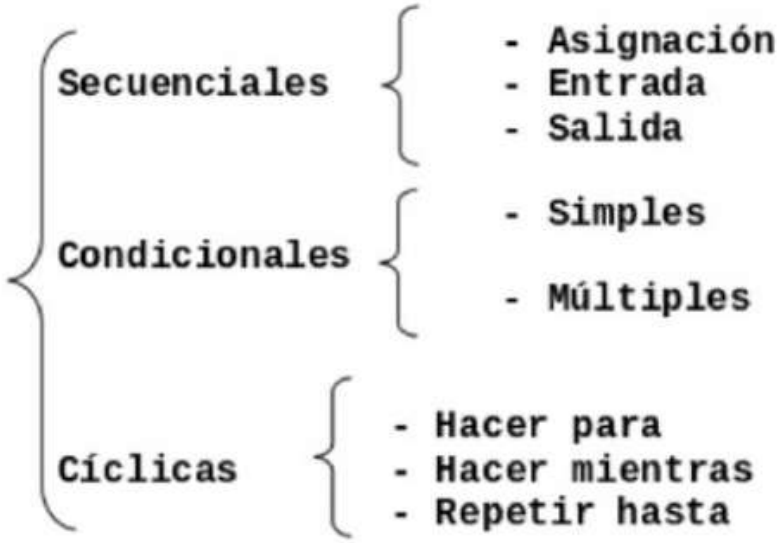
Viene a sustituir múltiples if anidados, y simplificar el proceso de análisis utilizando la estructura if.



5. ESTRUCTURAS DE CONTROL

- Otros de los aspectos importantes cuando hablamos de programación son las las estructuras de control, estas permiten modificar el flujo de ejecución de las instrucciones de un programa, es decir, permiten realizar una serie de operaciones que incrementan las posibilidades de un programa, sino existieran el código sería completamente líneas.

- Todos los lenguajes de programación presentan sus estructuras de control, similares entre ellas, añadiendo particularidades de funcionamiento en función del lenguaje escogido



5. ESTRUCTURAS DE CONTROL

5.1. Secuenciales

La estructura secuencial es aquella en la que una acción (instrucción) sigue a otra en secuencia. Las tareas se suceden de tal modo que la salida de una es la entrada de la siguiente y así sucesivamente hasta el fin del proceso.

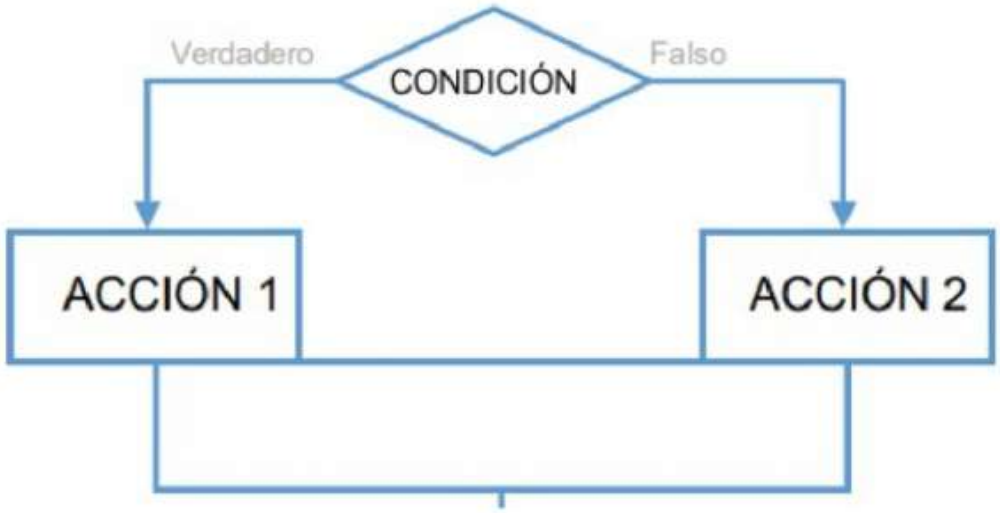
```
INICIO
  Accion1
  Accion2
  .
  .
  AccionN
FIN
```

En Diagrama de flujo se realiza así:

5. ESTRUCTURAS DE CONTROL

5.2. Condicionales

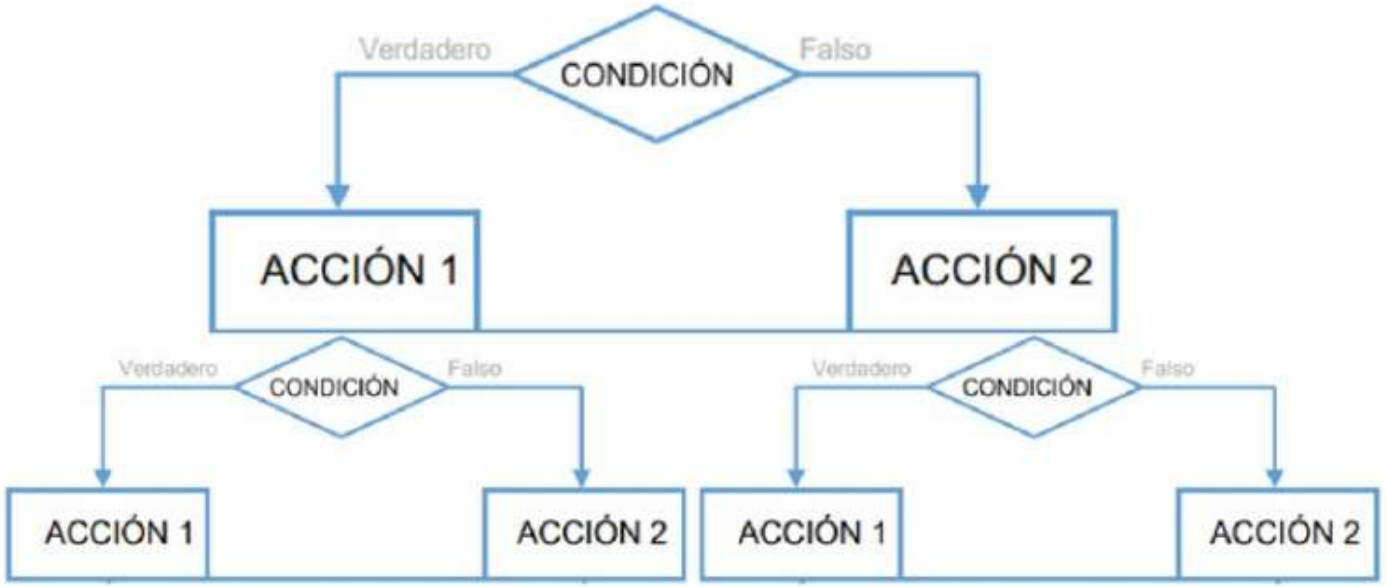
if-else. Se ejecuta un grupo u otro de sentencias en función de la condición o condiciones evaluadas en esta estructura.



5. ESTRUCTURAS DE CONTROL

5.2. Condicionales

if-else-if. Se ejecuta un grupo u otro de sentencias en función de la condición o condiciones evaluadas en esta estructura.



5. ESTRUCTURAS DE CONTROL

5.3. Cíclicas

Las estructuras cíclicas, permite al equipo realizar instrucciones de forma cíclica dependiendo del tipo de estructura que es, mientras se cumpla una condición, durante un número determinado de veces, hasta que...

while. Ejecutar un grupo de sentencias mientras se cumpla una condición.

5. ESTRUCTURAS DE CONTROL

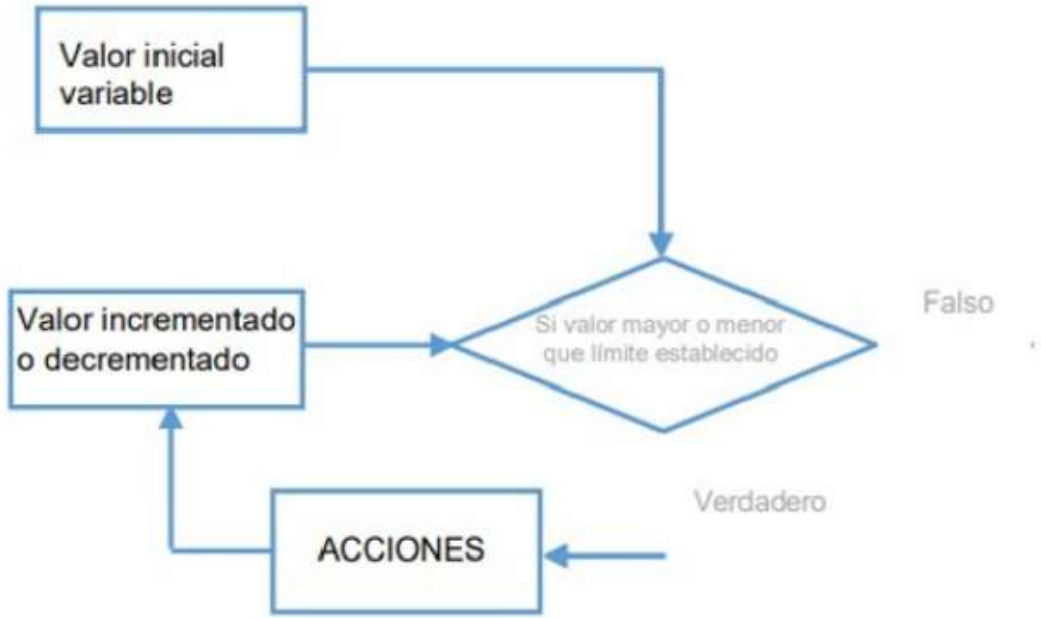
5.3. Cíclicas

do-mientras. Ejecutar un grupo de sentencias hasta que se cumpla una condición. Similar en estructura a la anterior, cambia la evaluación de los booleanos

5. ESTRUCTURAS DE CONTROL

5.3. Cíclicas

for. Ejecutar un grupo de sentencias un número determinado de veces,, podemos llamarlo desde-hasta que.



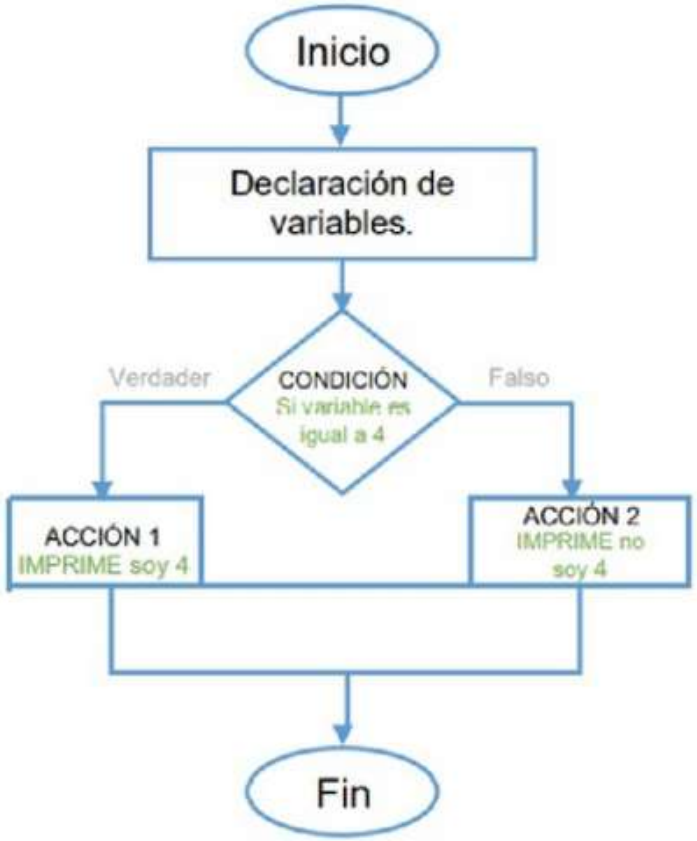
6. FUNCIONES

- Cuando hablamos de funciones en programación, estamos refiriéndonos a una subrutina o subprograma, que forma parte del programa principal.

- Cada una de estas funciones recibe un nombre que la identifica entre el resto de subrutinas, y que permite que sea invocada desde cualquier parte del código total del programa.

- Opcionalmente puede recibir valores, se ejecuta y puede devolver un valor. La existencia de las funciones permite obtener un programa total más ordenado.

Nota: La utilidad de las funciones, es la posibilidad de agrupar instrucciones y tener la capacidad de utilizar estas instrucciones cuando sea necesario.



6. FUNCIONES

Imaginemos que dentro de un mismo programa principal, se pide al usuario que introduce un valor por teclado, entre otras operaciones, ¿sería necesario realizar la comprobación de la imagen anterior cada vez que se reciba una variable, o bastará con invocar a esta subrutina?

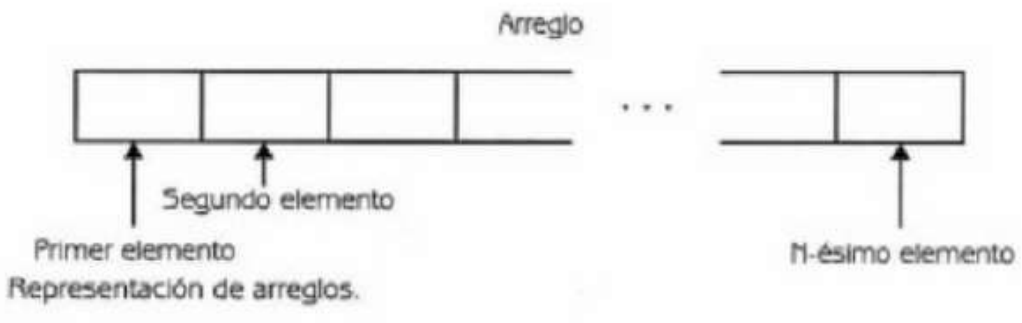
Si la opción escogida es la segunda, solo tendremos que introducir una vez el anterior fragmento de programa, por lo que tardaremos menos en obtener el programa final, además el resultado del código siempre será más legible, y se disminuirá la posibilidad de cometer algún error al escribirlo, en el caso de hacerlo, también será más cómodo depurar el código.

Por todas estas razones, conviene evitar que nuestro programa contenga código repetitivo.

7. ARRAYS(ARREGLOS)

Un arreglo se define como una colección finita, homogénea y ordenada de elementos.

- **Finita**: todo arreglo tiene un límite, es decirse debe determinar cuál será el número máximo de elementos que podrán formar parte del arreglo
- **Homogénea**: todos los elementos de un arreglo son del mismo tipo (todos enteros, todos reales, etc., pero nunca una combinación de distintos tipos).
- **Ordenada**: se puede determinar cuál es el primer elemento, el segundo, el tercero,... y el n-ésimo elemento.



Representación de arreglos.

7. ARRAYS(ARREGLOS)

Un arreglo tiene la característica de que puede almacenar a N elementos del mismo tipo y además permite el acceso a cada uno de estos elementos. Así, se distinguen dos partes en los arreglos:

- Los componentes.

- Los índices. Los componentes hacen referencia a los elementos que componen o forman el arreglo. Es decir, son los valores que se almacenan en cada una de sus casillas. Los índices, por otra parte, son los que permiten accesar a los componentes del arreglo en forma individual. Para hacer referencia a un componente de un arreglo se necesita:

    - El nombre del arreglo.

    - El índice del elemento.

7. ARRAYS(ARREGLOS)

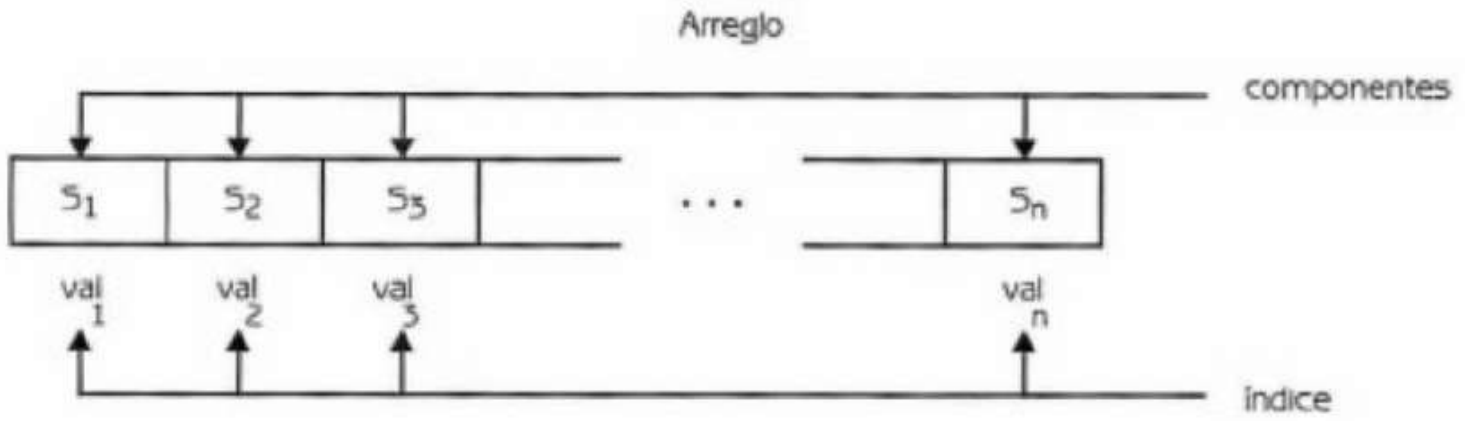


Figura 4.2 Indices y componentes de un arreglo.

7. ARRAYS(ARREGLOS)

Cuando se declara un array debemos especificar el tipo de datos para todos los elementos del arreglo. El tipo de los elementos no tiene que ser necesariamente el mismo que el de los índices.

Observaciones:

- El tipo del índice puede ser cualquier tipo ordinal (caracter, entero, etc.).
- El tipo de los componentes puede ser cualquier tipo (entero, real, cadena de caracteres, registro, arreglo, etc.).
- Se utilizan los corchetes "[]" para indicar el índice de un arreglo. Entre los [] se debe escribir un valor ordinal (puede ser una variable, una constante o una expresión tan compleja como se quiera, pero que dé como resultado un valor ordinal).

7. ARRAYS(ARREGLOS)

Ejemplo 4.3

Sea ARRE un arreglo de 70 elementos enteros con índices enteros. Su representación queda como se muestra en la figura 4.3.

ARRE = ARREGLO [1. .70] DE enteros

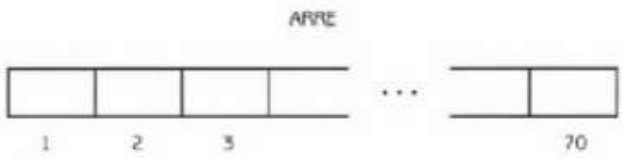


Figura 4.3

- NTE = (70 – 1 + 1) = 70
- Cada elemento del arreglo ARRE será un número entero y podrá accesarse por medio de un índice que será un valor comprendido entre 1 y 70.

Así por ejemplo:

ARRE[1] hace referencia al elemento de la posición 1.
ARRE[2] hace referencia al elemento de la posición 2.
. . .
. . .
ARRE[70]hace referencia al elemento de la posición 70.

7. ARRAYS(ARREGLOS)

Con los Arrays podemos realizar las siguientes operaciones:

- Lectura/Escritura.
- Asignación.
- Actualización:
    - Inserción.
    - Eliminación.
    - Modificación.

Leer ARRE[1],
Leer ARRE[2],
...
Leer ARRE[70]

ARRE[1], ARRE[2], ... ,ARRE[70]

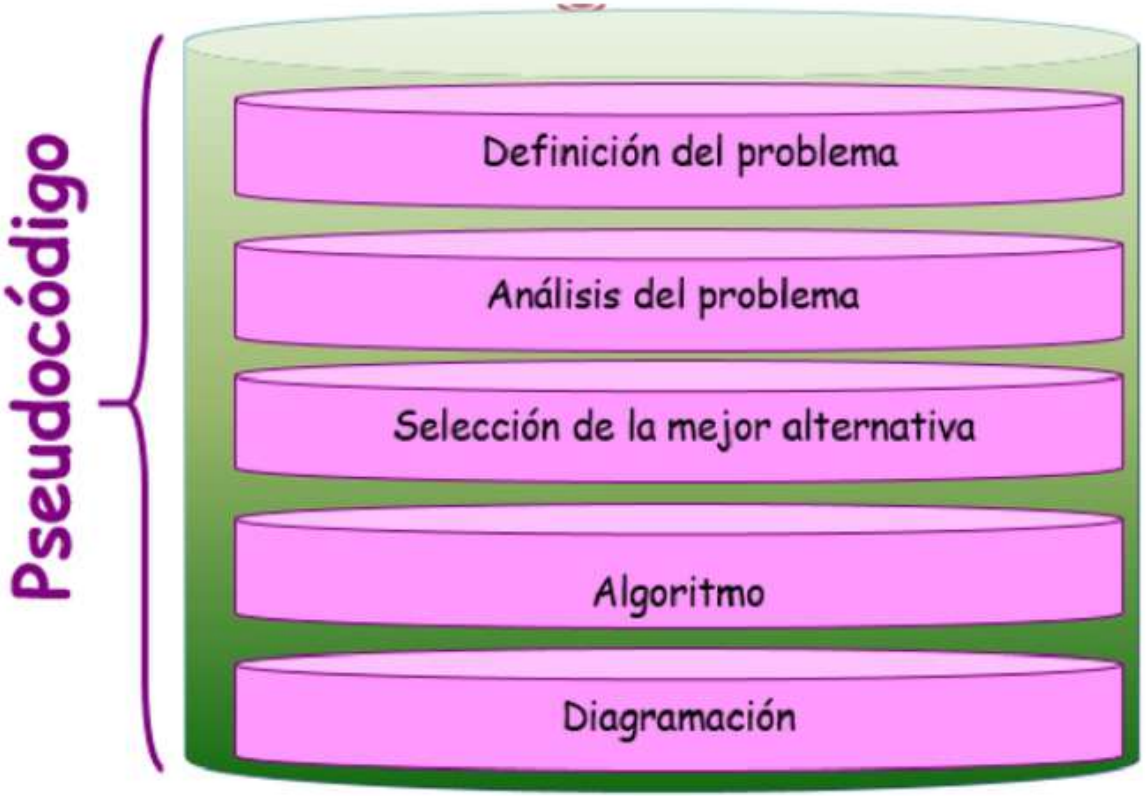
De esta forma no resulta práctico, por lo tanto se usará un ciclo para leer todos los elementos del arreglo.

```
Hacer I ← 1
Repetir con I desde 1 hasta 70
        Leer ARRE[I]
        Hacer I ← I + 1
{Fin del ciclo}
```

- Ordenación.

- Búsqueda. Como los arreglos son datos estructurados, muchas de estas operaciones no pueden llevarse a cabo de manera global, sino que se debe trabajar sobre cada elemento. A continuación se analizará cada una de estas operaciones. Ordenación y búsqueda, serán analizadas en los problemas que presentaremos posteriormente.

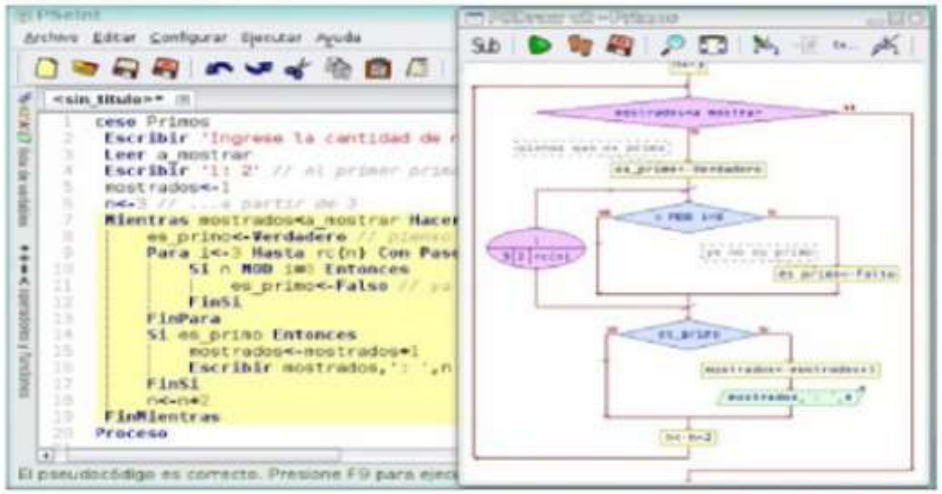
8. PSEUDOCÓDIGOS



8. PSEUDOCÓDIGO

- Permite con cierta facilidad la realización de futuras correcciones o actualizaciones gracias a que no es un sistema de representación rígido.

- Permite obtener la solución de un problema mediante aproximaciones sucesivas, es decir, lo que se conoce comúnmente como diseño descendente o **Top down** (Programación Modular) y que consiste en la descomposición sucesiva del problema en niveles más pequeños, lo que nos permite la simplificación del problema general.



Aplicación PseInt

8. PSEUDOCÓDIGO

Comentarios en pseudocódigo

Permite describir, explicar notas hechas por el programador desde un renglón hasta más de uno. Cuando el pseudocódigo es traducido al lenguaje de programación, el comentario es ignorado por la máquina. La simbología utilizada depende del número de renglones si sólo es uno; debe ponerse " // " al principio de la oración, pero cuando es más de un renglón de usa " /* " al principio y al final (para indicar que el comentario ha terminado) se pone un " */ ".

Las variables en pseudocódigo

Las variables son representación de un espacio de la memoria que guarda un valor que será utilizado para algún proceso, donde dicho valor puede ser modificado durante la ejecución. Están compuestas generalmente por un identificador (nombre que es asignado, donde no puede coincidir con las palabras reservadas, deben comenzar con una letra o guion evitando usar tilde y Ñ o ñ, no debe empezar con número ni espacio, pero si pueden ponerse números después de la primera letra.

Dentro de las variables encontramos a los tipos de datos simples (primitivos). Al declarar una variable debemos indicar que tipo de dato es.

8. PSEUDOCÓDIGO

Constantes en pseudocódigo:

Representa un espacio de memoria que guarda un valor definido y servirá para algún proceso durante el programa, dicho valor es fijo (no se puede modificar).

La ejecución de un programa consiste en la realización secuencial del conjunto de instrucciones de que se compone. Las instrucciones de un programa consisten en general en modificaciones sobre los objetos del programa, desde un estado inicial hasta otro final.

Las partes principales de un programa serían tres:

- Entrada de datos (desde los dispositivos externos hasta memoria central).
- Proceso (paso de un estado inicial a un estado final).
- Obtención de resultados (desde memoria central hacia los dispositivos externos).

8. PSEUDOCÓDIGO

Una instrucción se caracteriza por un **estado inicial**, que es el estado de los objetos antes de la ejecución de la instrucción, y otro **estado final** que es en el que quedan los objetos después de la ejecución de la misma.

Instrucciones de declaración.

Anuncian la utilización de objetos en un programa indicando qué identificador, tipo y otras características corresponde a cada uno de ellos.

Instrucciones de entrada.

Su misión consiste en tomar uno o varios datos desde un dispositivo de entrada y almacenarlos en la Memoria Central, en los objetos cuyos identificadores aparecen en la propia instrucción. Su sintaxis metodológica es:

8. PSEUDOCÓDIGO

Instrucciones de salida.

Es el conjunto de instrucciones que muestran el valor de algunos objetos en los dispositivos de salida (monitor...). Sintaxis:

escribir <lista de objetos>

Instrucciones de asignación.

Permiten realizar cálculos evaluando una expresión y depositando su valor final en un objeto o realizar movimientos de datos de un objeto a otro. Su sintaxis es:

variable expresión

Ejemplo.- EDAD EDAD_MAX

Una expresión puede ser un valor, una variable, una constante, una función o una combinación de todo lo anterior empleando operadores.

Ejemplo.- EDAD 17 * EDAD

Esta instrucción **se realiza en dos tiempos**; primero se evalúa la expresión convirtiéndose en su valor final; segundo, el valor final se asigna al objeto borrándose el valor previo que éste pudiese tener.

El objeto y la expresión deben coincidir en tipo y se admite que el propio objeto que recibe el valor final de la expresión pueda intervenir en la misma, pero entendiéndose que lo hace con su valor anterior.

8. PSEUDOCÓDIGO

•**Instrucciones de control.**

Son instrucciones que tienen como objetivo el controlar la ejecución de otras instrucciones o alterar el orden de ejecución normal de las mismas.

•**Instrucciones alternativas:**

Controlan la ejecución de uno o varios bloques de instrucciones dependiendo del cumplimiento o no de alguna condición o del valor final de una expresión.

Alternativa simple. Controla la ejecución de un conjunto de instrucciones por el cumplimiento o no de una condición.
Sintaxis: **si**
CONDICION **entonces**
.....
Finsi

Alternativa doble. Controla la ejecución de dos conjuntos de instrucciones por el cumplimiento o no de una condición. Si se cumple, se ejecutan las instrucciones del primer bloque y si no se cumple, las del segundo. Es una ampliación del anterior.

si CONDICION **entonces**
...
sino
...
finsi

8. PSEUDOCÓDIGO

Alternativa múltiple: Controla la ejecución de varios conjuntos de instrucciones por el valor final de una expresión, de tal forma que cada conjunto de instrucciones está ligado a un posible valor de la expresión. Se ejecutará el conjunto que se encuentre relacionado con el valor que resulte de la evaluación de la expresión.

Sintaxis:

opción EXPRESION **de**
V1 **hacer** Ins1, Ins2....
V2 **hacer** Ins1, Ins2...
....
VN **hacer** Ins1, Ins2...
Otra hacer ....
Fin opción

Ejemplo.-

opción EDAD de
4 hacer regalo ← "JUGUETE"
15 hacer regalo ← "CD"
otro hacer regalo ← "LIBRO"
Fin opción

8. PSEUDOCÓDIGO

- **Instrucciones repetitivas:** Son aquellas que controlan la repetición de un conjunto de instrucciones denominado rango mediante la evaluación de una condición que se realiza cada nueva repetición o por medio de un *contador* asociado.

  - Instrucción **MIENTRAS.** El conjunto de instrucciones que configuran su rango se ejecutan mientras se cumpla la condición que será evaluada siempre antes de cada repetición, es decir, mientras la condición sea CIERTA.

mientras CONDICION **hacer**
....
Fin mientras

Ejemplo.-

```
leer numero, potencia
resultado  1
mientras (potencia > 0) hacer
        resultado  resultado * numero
        potencia  potencia – 1
finmientras
escribir resultado
```

8. PSEUDOCÓDIGO

- Instrucción **REPETIR.** Controla la ejecución del conjunto de instrucciones que configuran su rango de tal forma que éstas se ejecutan hasta que se cumpla la condición, que será evaluada siempre después de cada repetición, es decir, hasta que la condición sea CIERTA.

repetir Instr
...
hasta CONDICION

Ejemplo.- El ejemplo anterior pero con estructura Repetir sería

```
repetir
        resultado  resultado * numero
        potencia  potencia – 1
hasta (potencia = 0)
```

8. PSEUDOCÓDIGO

- Instrucción **PARA**. Controla la ejecución del conjunto de instrucciones de su rango de tal forma que éstas se ejecutan un número determinado de veces que queda definido en lo que se denomina cabecera del bucle. En la cabecera se define una variable de control del bucle, su valor inicial, su valor final y su incremento o decremento. Su sintaxis:

para Vc **de** Vi **a** Vf **con incremento** J **hacer**
....
....

finpara

Ejemplo.-
para H de 1 a potencia con incremento 1 hace
resultado resultado * numero

Fin para

Con esta clase de bucle nos ahorramos instrucción de incremento al estar especificada en la cabecera del bucle. Es ideal para repeticiones con incrementos fijos (positivos o negativos).

8. PSEUDOCÓDIGO

Siempre que queremos escribir un programa en PSeInt iniciamos debajo de la palabra

Proceso sin_titulo
//escribimos el cuerpo del programa;

FinProceso
Para poner un comentario en el programa se utiliza **//.**
La palabra reservada **Escribir** escribe en la pantalla lo que esta encerrado entre comillas. En sintaxis flexible también podemos utilizar la palabra **Imprimir** o **Mostrar.**
Concatenar texto

```
Proceso concatenar
Escribir "Mi primer programa ";
Escribir " en PSeInt ";
FinProceso
```

La salida del programa es
Mi primer programa
en PSeInt
En sintaxis estricta, las sentencias siempre finalizan en punto y coma.

8. PSEUDOCÓDIGO

Declarar variables

En sintaxis estricta, siempre que necesitemos hacer un programa, tendremos que declarar variables para poder guardar la información que introduzcamos al programa.
Los tipos de datos básicos soportados son los siguientes:

- Entero: solo números enteros.
- Real: números con cifras decimales.
- Carácter: cuando queremos guardar un carácter.
- Lógico: cuando necesitamos guardar una expresión lógica (verdadero o falso)
- Cadena: cuando queremos guardar cadenas de caracteres.

Nota: *Cadena y Carácter son términos equivalentes, no genera error que las escribamos indistintamente*
Ejemplos: *Si queremos declarar una variable de tipo entero sería así:*

```
Definir  numero Como Entero;
```

8. PSEUDOCÓDIGO

PSeInt proporciona los siguientes operadores:

Operador Función

() Agrupar expresiones
^ Operador para exponenciación
* Operador de multiplicación
/ Operador de división
% ó Mod Operador de cálculo de residuo
trunc(valor1 / valor2); Sintaxis de división entera
& ó y Operador lógica y
+ Operador de suma
- Operador de Resta
| ó o Operador lógico o

Nota: *En sintaxis flexible, podemos utilizar también los operadores & | y mod como y o y % respectivamente.*

8. PSEUDOCÓDIGO

Leer valores y almacenarlos en las variables

Cuando nosotros queremos leer un valor y almacenarlo en una variables usaremos la palabra ***Leer < variable>;*** y cuando queremos asignar un valor o una operación matemática usaremos <- que es el símbolo de < mas - .

Ejemplo sobre lectura de datos

```
Proceso lectura_datos
Definir nombre Como Cadena;
Escribir "Ingrese su nombre ";
Leer nombre
Escribir "Bienvenido " sin saltar;
Escribir nombre
FinProceso
```

8. PSEUDOCÓDIGO

Los programas pueden toman sus propias decisiones, en PSeInt existen instrucciones condicionales que se describen a continuación:

Instrucción Si: Sintaxis
Si condición **Entonces**
instrucciones;
FinSi
ó
Si condición **Entonces**
instrucciones;
Sino
instrucciones;
FinSi

8. PSEUDOCÓDIGO

Esta se usa como sustituto en algunos casos del si anidado , por ser más práctico al aplicarlo en la evaluación de algunas condiciones.
Sintaxis

Segun variable **Hacer**
valor1, valor2, valor3, **... :** instrucciones;
valor4, valor5, valor6, ... : instrucciones;
[De Otro Modo : instrucciones;]

FinSegun

Los valores a evaluar, se separan por comas si hay varios, tal como aparece en la sintaxis valor1, valor2, etc., también se puede usar el sino que nos indica que en caso de no seleccionar ninguna de las instrucciones anteriores se ejecutan.
Nota importante: *En sintaxis estricta las opciones del Segun deben ser siempre del tipo numérico. Para poder evaluar opciones del tipo texto se debe personalizar el lenguaje utilizando sintaxis flexible en el editor.*

8. PSEUDOCÓDIGO

Usaremos una estructura Según. Para eso habilitamos sintaxis flexible yendo a personalizar lenguaje → Personalizar...
→ Utilizar sintaxis flexible

```
Proceso ejemplo_caso
Definir  zona Como Caracter;
Definir nombre Como Cadena;
Definir  ventas , comis , ihss,  tp Como Reales;
Escribir "Ingresar el nombre del empleado     ";
Leer  nombre;
Escribir "Ingresar las ventas   del empleado ";
Leer  ventas;
Escribir "Ingresar la zona de trabajo ";
Leer  zona;
Segun  Zona Hacer
'a','A'  :     comis<- 0.06  *  ventas;
'b','B'  :     comis<- 0.08  *  ventas;
'c','C'  :     comis<- 0.09  *  ventas;
De Otro Modo  : comis<- 0;
FinSegun
```

8. PSEUDOCÓDIGO

El operador | (O) se utiliza cuando estamos evaluando dos o más condiciones y queremos que la condición se cumpla cuando una de las condiciones que estamos evaluando se hacen verdadera. Ejemplo
Cuando se introduce la zona en el ejercicio con la estructura Si solo evaluábamos una opción que la zona sea igual a la letra A y si el usuario escribía una a minúscula no2 se tomaba en cuenta pero esto se puede corregir de esta manera :

```
Si zona ='A'   |   zona ='a'
Entonces
comis<- 0.06  *  ventas;
Sino Si zona='B' | zona='b' Entonces
comis<- 0.08 * ventas;
Sino si zona='C' | zona='c' Entonces
comis<- 0.09 * ventas;
Sino
comis<- 0;
FinSi
FinSi
FinSi
```

Ahora la condición dice, ***si zona es igual a la letra A o es igual a la letra a***, cualquiera que sea la zona a o A en ambos casos la condición es verdadera, ahora el usuario puede usar mayúsculas y minúsculas y el resultado será el mismo.

8. PSEUDOCÓDIGO

El operador Y (&) se utiliza cuando estamos evaluando dos o más condiciones y queremos que la condición se cumpla cuando las dos condiciones que estamos evaluando se hacen verdadera. Ejemplo

Ejemplo sobre el operador &

Se ingresa un número y se desea saber si dicho número está entre 50 y 100.

```
Proceso ejemplo_operador_y
Definir num  Como Entero;
Escribir "Número a evaluar";
Leer num;
Si num >=50  & num<=100 Entonces
Escribir " El número está entre 50 y 100";
Sino
Escribir " Fuera del rango 50 y 100";
FinSi
FinProceso
```

8. PSEUDOCÓDIGO

Sintaxis

Mientras condición Hacer
instrucciones;
FinMientras

El ciclo mientras se utiliza cuando se quiere ejecutar repetidamente un bloque instrucciones basado en una condición, el ciclo se repite mientras la condición se cumple.

Ejemplo.

Ingresar 10 nombres

```
Proceso contador
Definir contador Como Entero;
Definir nombre Como Cadena;
Contador<-0;
Mientras contador<10 Hacer
Escribir "Ingresar el nombre";
Leer nombre;
contador<- contador + 1;
FinMientras
FinProceso
```

En este programa introducimos el concepto de contador , que es una variable que se incrementa su valor en 1 y de esta manera contamos cuantos nombres se van ingresando para parar cuando ingresemos 10 , cuando el contador vale 10 la condición de contador < 10 ya no se cumple porque es igual y el ciclo termina.

8. PSEUDOCÓDIGO

Sintaxis
Para variable <- valor_inicial Hasta valor_final Con Paso Paso Hacer
instrucciones
FinPara

Descripción
El **ciclo Para** se utiliza generalmente para ejecutar un conjunto de instrucciones que se repiten un número de veces establecido antes de ejecutar el ciclo. **Variable :** es de tipo entero
Valor_inicial : este puede se un número entero o una variable entera. **Valor_final :** este puede se un número entero o una variable entera.
Paso : este puede se un número entero o una variable entera.
***Nota**: el paso 1 puede omitirse, tanto en sintaxis estricta como flexible*
Ejemplo : presentar los números del 1 al 10 en la pantalla.

```
Proceso ciclo_Para
Definir I Como Entero;
Para  I<-1  Hasta  10  Con Paso 1 Hacer
Escribir  I;
FinPara
FinProceso
```

Ciclos negativos
PSeInt también puede realizar ciclos negativos para mostrar, por ejemplo secuencias de mayor a menor, solamente invirtiendo el orden de los números del ejercicio anterior y colocando como Paso -1

```
Proceso ciclo_Para_negativo
Definir I Como Entero;
Para  I<-10  Hasta  1  Con Paso -1 Hacer //En ciclos negativos el paso no puede omitirse.
Escribir  I;
FinPara
FinProceso
```

8. PSEUDOCÓDIGO

Sintaxis:
Repetir
Instrucciones;
Hasta Que condición

Descripción
El ciclo Repetir es lo contrario al ciclo Mientras, en éste la ejecución se lleva a cabo hasta que se cumple la condición impuesta.
La diferencia con el ciclo Mientras radica en que este evalúa la condición desde el principio, y si está no se cumple, el código que está encerrado dentro del cuerpo del mientras no se ejecuta.
En cambio, el Repetir - Mientras Que evalúa la condición para seguir ejecutándose luego de haber ejecutado el código dentro de su cuerpo, es decir siempre se ejecuta por lo menos una vez el código.
***Nota**: En perfil flexible, habilitando sintaxis flexible o en personalizar también es posible usar la estructura*
Hacer
//Instrucciones;
Mientras Que
o
Repetir
//Instrucciones;
Mientras Que

8. EJEMPLO

Ingresar el nombre del alumno, la nota , luego preguntar si desea continuar , al final presentar el número de aprobados y suspensos.

```
Proceso ejemplo_repetir
Definir resp Como Caracter;
Definir nota Como Real;
Definir ap,sus Como Enteros;
Definir nombre como Cadena;
ap<-0;
sus<-0;
Repetir
Escribir "ingresar el nombre del alumno ";
Leer nombre;
Escribir "ingresar la nota del alumno ";
Leer nota;
Si nota >= 5 Entonces
ap<-ap+1;
Sino
sus<-sus+1;
FinSi
Escribir " Desea continuar S/N";
Leer resp;
Hasta Que resp='n'  |  resp='N';
Escribir " Aprobados ",ap;
Escribir " Suspensos ",sus;
FinProceso
```

RESUMEN DE LA UNIDAD

- En esta unidad se abordan las diferentes estructuras que se necesitan en la metodología de la programación.
- Hay tres tipos principales de estructuras: secuenciales, condicionales y cíclicas.
- Los pseudocódigos son el estándar para diseñar nuestros primeros fragmentos de código.
- Una vez que se crea el pseudocódigo, el siguiente paso es transcribir las instrucciones al lenguaje de programación que deseamos utilizar.

RECURSOS DE INTERÉS

- **Lntroducción a Java Espacio de Trabajo de:** https://openwebinars.net/blog/que-es-java/
- **Pseint:**https://pseint.sourceforge.net/
- **Java Oficial web site:** https://www.java.com/es/
- **IMB** https://www.ibm.com/es-es/topics/java

BIBLIOGRAFÍA/WEBGRAFÍA

- Joyanes, Luis. "Metodología de la programación estructurada."
- Alonso Santos, María Dolores. "Programación estructurada."
- Enrique Gómez Jiménez ."Metodología de la programación: conceptos, lógica e implementación (Alfaomega)."
- Fernando Alonso Amo, José Crespo del Arco, Ángel Lucas González Martínez, Miguel Jiménez Gañán, Daniel Manrique Gamo. "Metodología de la programación."