

Introducción a la programación.

Contenidos:

- 1) Los paradigmas de programación
- 2) Tipos de datos
- 3) Representación de algoritmos
- 4) PSeInt

Los paradigmas de programación

Los estilos de programación han evolucionado con el tiempo, pasando por diferentes períodos:

- Programación desestructurada (entre 1950 y 1970)
- Programación estructurada clásica (entre 1970 y 1990)
- Programación modular (entre 1970 y 1990)
- Programación orientada a objetos (desde 1990)

¿Qué es un algoritmo?

Un **algoritmo** es una secuencia ordenada de pasos que resuelven un problema en tiempo finito. Principales características de los algoritmos:

- Contienen instrucciones concretas, sin ninguna ambigüedad.
- Deben terminar, es decir, son finitos.
- Todos sus pasos son simples y están ordenados.

Un **programa** es la traducción de un algoritmo a un lenguaje de programación capaz de ser entendido y procesado por el ordenador. Generalmente, el orden de las instrucciones de un programa es el mismo en que se escriben, pero muchas veces es necesario repetir conjuntos de instrucciones (bucles) o realizar saltos de una instrucción a otra.

Los paradigmas de programación

En la programación **desestructurada** clásica se usan bucles y saltos muy frecuentemente, dando lugar a un código farragoso, a veces confuso, que genera gran probabilidad de incurrir en errores y hace muy difícil el mantenimiento o evolución del programa. Los lenguajes más antiguos, como Fortran, Cobol, Basic, etc... se consideran desestructurados.

La programación **estructurada** consiste en la utilización de estructuras que optimizan los recursos lógicos y físicos del ordenador. Los lenguajes estructurados también se conocen como imperativos o de tercera generación. Por ejemplo, C, Pascal o Modula-2 son estructurados. Algunos lenguajes antiguos se adaptaron con el tiempo a este paradigma, aunque permitían seguir haciendo programación desestructurada si el programador lo deseaba.

La programación **modular** convive con la estructurada y con frecuencia se usa conjuntamente. Consiste en dividir el programa complejo en varios programas sencillos que interactúan entre ellos. Cada programa sencillo se llama "módulo". Los módulos deben ser independientes entre sí, no interferir unos con otros. El lenguaje modular clásico es "Modula-2".

La programación **orientada a objetos** (OOP / POO) es una evolución de la anterior. Es programación estructurada y modular al mismo tiempo, en la que las instrucciones y los datos se encapsulan en entidades denominadas "clases", de las que luego se crean los "objetos", que tienen una serie de propiedades y capacidades para realizar acciones o "métodos". Lo veremos más adelante.

Evolución histórica de los lenguajes de programación

Con el paso del tiempo, se va incrementando el nivel de abstracción, pero en la práctica, los de una generación no terminan de sustituir a los de la anterior:

Lenguajes de primera generación (1GL): Código máquina.

Lenguajes de segunda generación (2GL): Lenguajes ensamblador, por ejemplo, COBOL 1959.

Lenguajes de tercera generación (3GL): La mayoría de los lenguajes modernos, diseñados para facilitar la programación a los humanos. Ejemplos: C, Java.

Lenguajes de cuarta generación (4GL): se ha dado este nombre a ciertas herramientas que permiten construir aplicaciones sencillas combinando piezas prefabricadas. Hoy se piensa que estas herramientas no son, propiamente hablando, lenguajes. Algunos proponen reservar el nombre de cuarta generación para la programación orientada a objetos.

Lenguajes de quinta generación (5GL): La intención es que el programador establezca qué problema ha de ser resuelto y las condiciones a reunir, y la máquina lo resuelve. Se usan en inteligencia artificial. Ejemplo: Prolog.

Tipos de datos

Los programas manejan **datos**. Un tipo de datos es la propiedad de un valor que determina su dominio (qué valores puede tomar), qué operaciones se le pueden aplicar y cómo es representado internamente por el ordenador. Es decir, los datos se caracterizan por:

- **Dominio de posibles valores:** qué valores puede tomar.
- **Cómo se representan:** cuál es la representación interna y cómo se representan en el lenguaje de programación.
- **Operadores asociados:** qué operaciones o cálculos se pueden realizar con esos datos.

Todos los valores que aparecen en un programa tienen un **tipo**. Los tipos elementales o primitivos de un lenguaje de programación son:

- Entero (byte, short, int, long)
- Booleano (boolean)
- Carácter (char)
- Real (float, double)

Tipos de datos

Números enteros

El tipo `int` (del inglés `integer`, que significa «entero») permite representar números enteros. Los valores que puede tomar un `int` son todos los números enteros: ... -3, -2, -1, 0, 1, 2, 3, ...

Los números enteros literales se escriben con un signo opcional seguido por una secuencia de dígitos:

1570

+4591

-12

Tipos de datos

Números reales

El tipo float permite representar números reales. El nombre float viene del término "punto flotante", que es la manera en que el computador representa internamente los números reales. Hay que tener mucho cuidado, porque los números reales no se pueden representar de manera exacta en un computador. Por ejemplo, el número decimal 0.7 es representado internamente por el computador mediante la aproximación 0.69999999999999996. Todas las operaciones entre valores float son aproximaciones.

Los números reales literales se escriben separando la parte entera de la decimal con un punto. Las partes entera y decimal pueden ser omitidas si alguna de ellas es cero:

>>> 881.9843000	>>> -3.14159	>>> 1024.	>>> .22
881.9843	-3.14159	1024.0	0.22

Otra representación es la notación científica, en la que se escribe un factor y una potencia de diez separados por una letra e. Por ejemplo:

```
>>> -2.45E4
-24500.0
>>> 7e-2
```


Tipos de datos

Valores lógicos

Los valores lógicos True y False (verdadero y falso) son de tipo bool, que representa valores lógicos. El nombre bool viene del matemático George Boole, quien creó un sistema algebraico para la lógica binaria. Por lo mismo, a True y False también se les llama valores booleanos.

Tipos de datos

Texto

A los valores que representan texto se les llama strings, y tienen el tipo str. Los strings literales pueden ser representados con texto entre comillas simples o comillas dobles:

"ejemplo 1"

'ejemplo 2'

La ventaja de tener dos tipos de comillas es que se puede usar uno de ellos cuando el otro aparece como parte del texto:

"Let's go!"

'Ella dijo "hola"'

Tipos de datos

Es importante entender que los strings no son lo mismo que los valores que en él pueden estar representados:

```
>>> 5 == '5' => False
```

```
>>> True == 'True' => False
```

Los strings que difieren en mayúsculas y minúsculas, o en espacios también son distintos:

```
>>> 'mesa' == 'Mesa' => False
```

```
>>> ' mesa' == 'mesa ' => False
```

Nulo

Existe un valor llamado None (en inglés, «ninguno») que es utilizado para representar casos en que ningún valor es válido, o para indicar que una variable todavía no tiene un valor que tenga sentido.

Representación de algoritmos: El pseudocódigo

Para la creación de programas, como paso previo a la utilización de lenguajes de programación, usamos lo que se conoce como "pseudocódigo", es decir, un lenguaje de especificación de algoritmos que usa una notación en lenguaje natural, permitiendo representar la programación estructurada y haciendo que el paso final a la codificación de un programa sea relativamente fácil.

El pseudocódigo debe cumplir estas reglas:

- Debe ser **preciso** e indicar el orden de realización de cada paso.
- Debe estar **definido**, es decir, debe dar un mismo resultado (si ponemos los mismos datos no puede dar un resultado distinto).
- Debe ser **finito**, es decir, debe de acabar en algún punto.
- Debe ser **independiente** de un lenguaje de programación


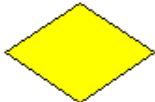








Representación de algoritmos: Los diagramas de flujo

Un diagrama de flujo es la representación gráfica de un algoritmo o proceso. Estos diagramas utilizan símbolos con significados definidos que representan los pasos del algoritmo, y representan el flujo de ejecución mediante flechas que conectan los puntos de inicio y de fin del proceso.

Los diagramas de flujo utilizan una simbología y normas determinadas. Algunos de los símbolos más utilizados son:

- Óvalo o Elipse: Inicio y Final (Abre y cierra el diagrama).
- Rectángulo: Actividad (Representa la ejecución de una o más actividades o procedimientos).
- Rombo: Decisión (Formula una pregunta o cuestión).
- Círculo: Conector (Representa el enlace de actividades con otra dentro de un procedimiento).
- Triángulo boca abajo: Archivo definitivo (Guarda un documento en forma permanente).
- Triángulo boca arriba: Archivo temporal (Proporciona un tiempo para el almacenamiento del documento).

Representación de algoritmos: Los diagramas de flujo

	Inicio/Final Se utiliza para indicar el inicio y el final de un diagrama; de inicio sólo puede salir una línea de flujo y al final sólo debe llegar una línea		Decisión Indica la comparación de dos datos y dependiendo del resultado lógico (falso o verdadero) se toma la decisión de seguir un camino del diagrama u otro
	Entrada/Salida Entrada/Salida de datos por cualquier dispositivo (scanner, lector de código de barras, micrófono, parlantes, etc.)		Impresora/Documento. Indica la presentación de uno o varios resultados en forma impresa
	Entrada por teclado. Entrada de datos por teclado. Indica que el computador debe esperar a que el usuario teclee un dato que se guardará en una variable o constante		Pantalla Instrucción de presentación de mensajes o resultados en pantalla
	Acción/Proceso Indica una acción o instrucción general que debe realizarse (operaciones aritméticas, asignaciones, etc.)		Conector Interno Indica el enlace de dos partes de un diagrama dentro de la misma página
	Flujo/Flechas de Dirección Indica el seguimiento lógico del diagrama. También indica el sentido de ejecución de las operaciones		Conector Externo Indica el enlace de dos partes de un diagrama en páginas diferentes

Pseudocódigo

```
1 Proceso sin_titulo
2
3   Escribir "Ingrese la cantidad: "; Leer "N";
4
5   Dimension a[n];
6   para i<-1 hasta 10 hacer
7     Escribir "Ingrese el dato" i;
8     Leer a[i];
9   FinPara
10
11 FinProceso
```

The screenshot shows the PSeInt application window. The main editor contains the pseudocode above. A tooltip is visible over line 5, stating: "(1) Las dimensiones deben ser constantes." The right sidebar shows a list of commands: Escribir, Leer, Asignar, Si-Entonces, Segun, Mientras, Repetir, and Para. The status bar at the bottom indicates the version v20120614.

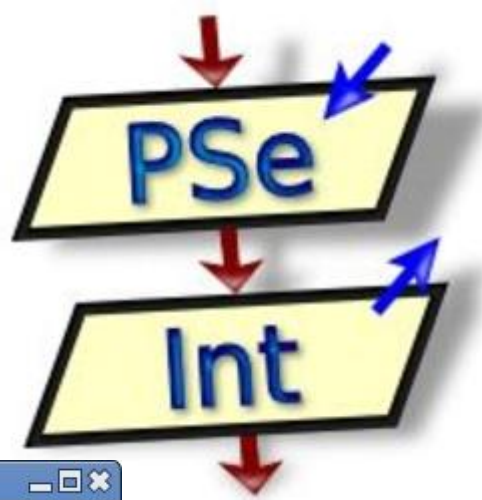
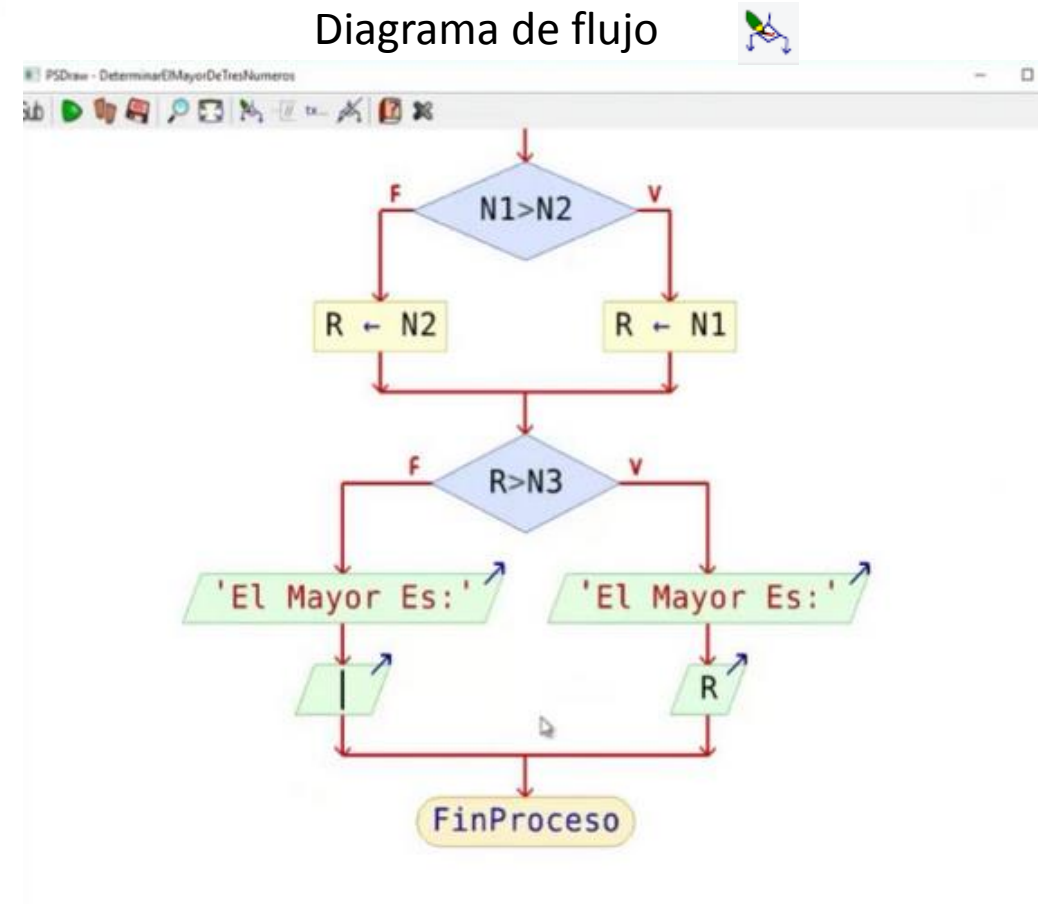


Diagrama de flujo



Pseudocódigo

```
Proceso sin_titulo
```

```
FinProceso
```

```
Proceso Ejemplo
```

```
FinProceso
```

```
+ Proceso Mal Nombre  
FinProceso
```

Diagrama de flujo

Proceso sin_titulo

FinProceso

Proceso Ejemplo

FinProceso

⊗ Proceso Mal Nombre

FinProceso

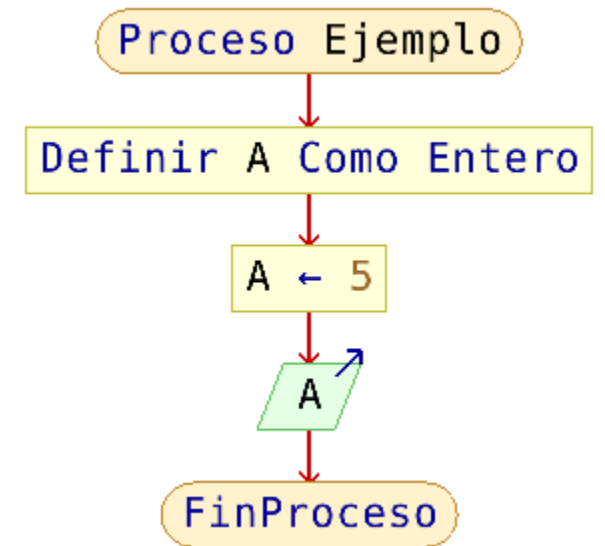
Operaciones: ASIGNACIÓN

Pseudocódigo

```
1  Proceso Ejemplo
2      Definir A como entero;
3      A <- 5;
4      Escribir A;
5  FinProceso
6
7  |
```



Diagrama de flujo



```
PSelnt - Ejecutando proceso EJEMPLO
*** Ejecución Iniciada. ***
5
*** Ejecución Finalizada. ***
```

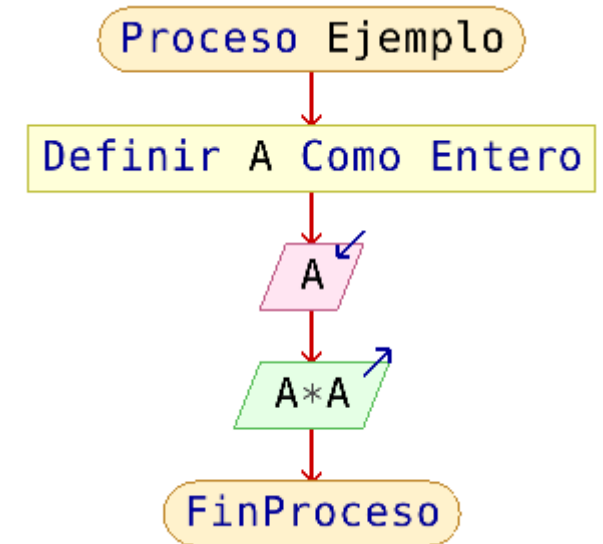
Operaciones: LECTURA Y ESCRITURA

Pseudocódigo

```
1  Proceso Ejemplo
2      Definir A como entero;
3      Leer A;
4      Escribir A*A;
5  FinProceso
6
```



Diagrama de flujo



```
PSeInt - Ejecutando proceso EJEMPLO
*** Ejecución Iniciada. ***
> 10
100
*** Ejecución Finalizada. ***
```

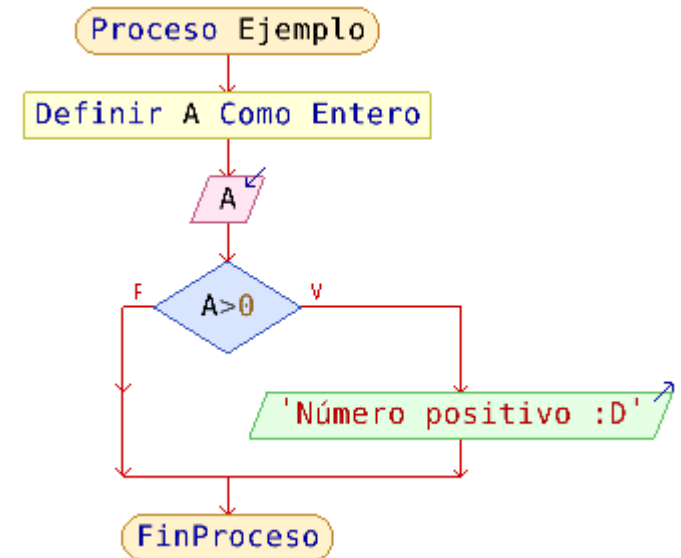
Operaciones: estructuras de selección

Pseudocódigo

```
Proceso Ejemplo
  Definir A Como Entero;
  Leer A;
  Si A>0 Entonces
    Escribir 'Número positivo :D';
  FinSi
FinProceso
```

```
*** Ejecución Iniciada. ***
> 0
*** Ejecución Finalizada. ***
```

Diagrama de flujo



```
*** Ejecución Iniciada. ***
> 10
Número positivo :D
*** Ejecución Finalizada. ***
```

Operaciones: estructuras de selección

Pseudocódigo

```
Proceso Ejemplo
  Definir A Como Entero;
  Leer A;
  Si A > 0 Entonces
    Escribir "Bonito número";
  SiNo
    Escribir "Menor o igual que 0 :(";
  FinSi
FinProceso
```

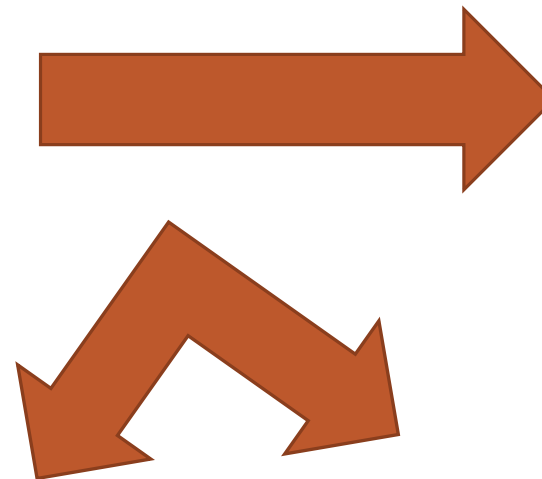
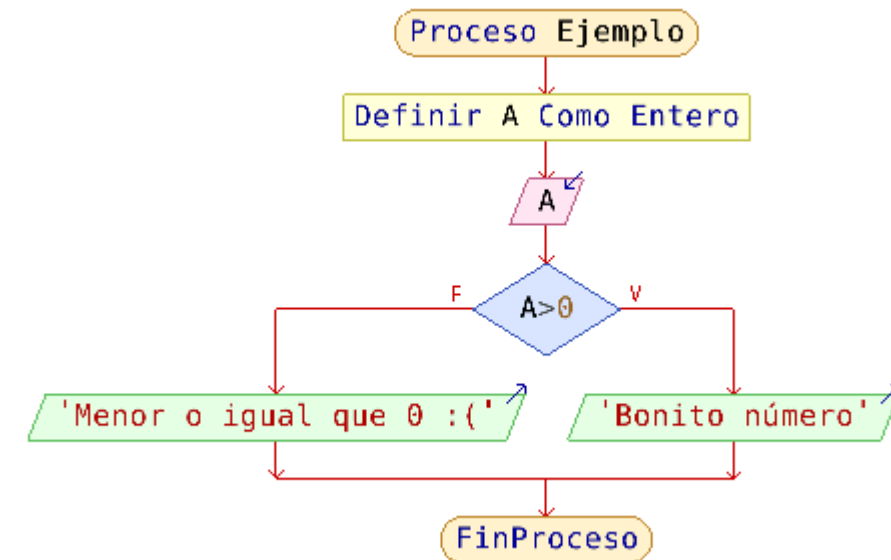


Diagrama de flujo



```
*** Ejecución Iniciada. ***
> 0
Menor o igual que 0 :(
*** Ejecución Finalizada. ***
```

```
*** Ejecución Iniciada. ***
> 5
Bonito número
*** Ejecución Finalizada. ***
```

Operaciones: estructuras de Control

Pseudocódigo

```
Proceso Ejemplo
  Definir A Como Entero;
  Leer A;
  Si A > 0 Entonces
    Escribir "Bonito número";
  SiNo
    Escribir "Menor o igual que 0 :(";
  FinSi
FinProceso
```

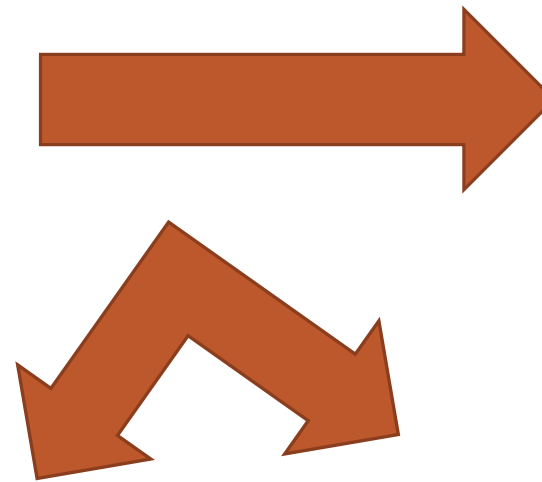
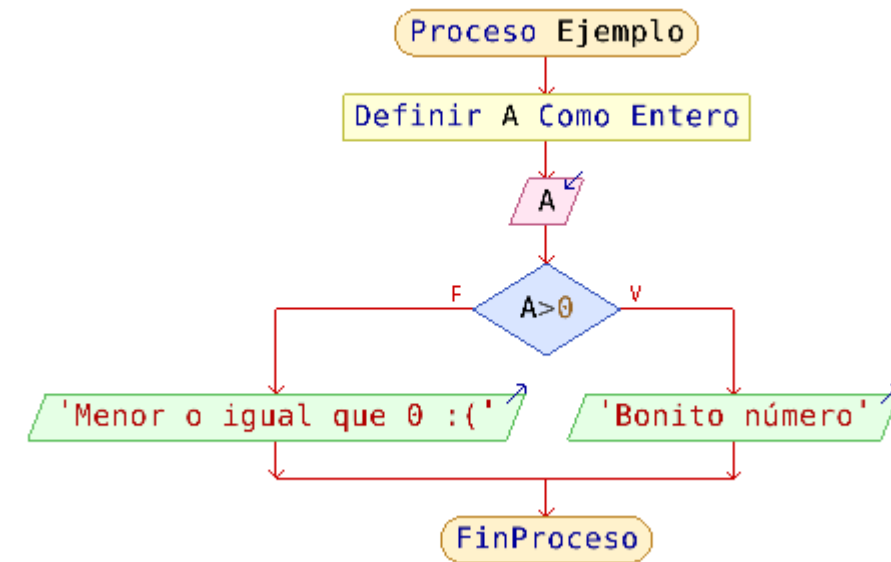


Diagrama de flujo



```
*** Ejecución Iniciada. ***
> 0
Menor o igual que 0 :(
*** Ejecución Finalizada. ***
```

```
*** Ejecución Iniciada. ***
> 5
Bonito número
*** Ejecución Finalizada. ***
```

Operaciones: estructuras de Control

Pseudocódigo

```
1  Proceso Ejemplo
2      Definir A Como Entero;
3      A ← 0;
4      Mientras A < 10 Hacer
5          Escribir A;
6          A ← A + 1;
7      FinMientras
8  FinProceso
```

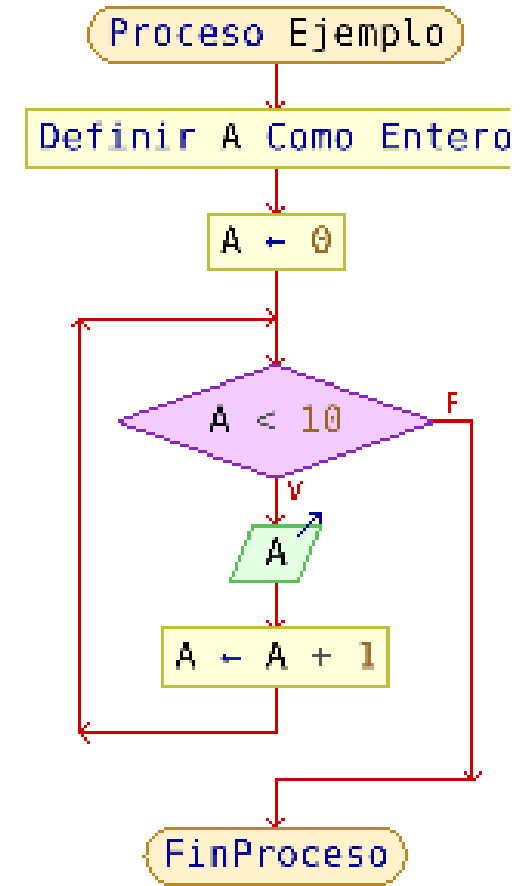


*** Ejecución Iniciada. ***

0
1
2
3
4
5
6
7
8
9

*** Ejecución Finalizada. ***

Diagrama de flujo



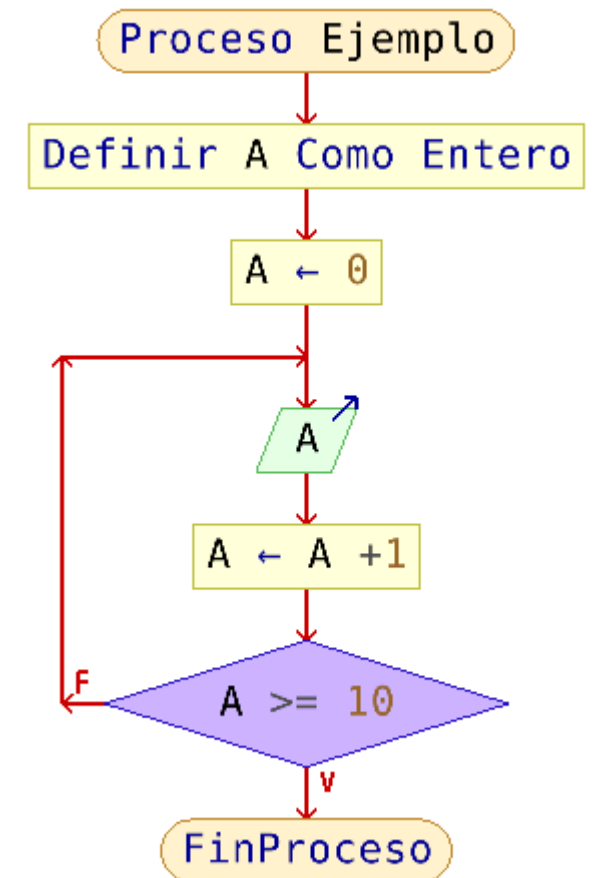
Operaciones: estructuras de Control

Pseudocódigo

```
Proceso Ejemplo
  Definir A Como Entero;
  A ← 0;
  Repetir
    Escribir A;
    A ← A + 1;
  Hasta Que A ≥ 10
FinProceso
```



Diagrama de flujo



*** Ejecución Iniciada. ***

0
1
2
3
4
5
6
7
8
9

*** Ejecución Finalizada. ***

Operaciones: estructuras de Control

Pseudocódigo

Diagrama de flujo

Proceso Ejemplo

Definir A Como Entero;

Para A<-0 Hasta 9 Hacer
Escribir A;

FinPara

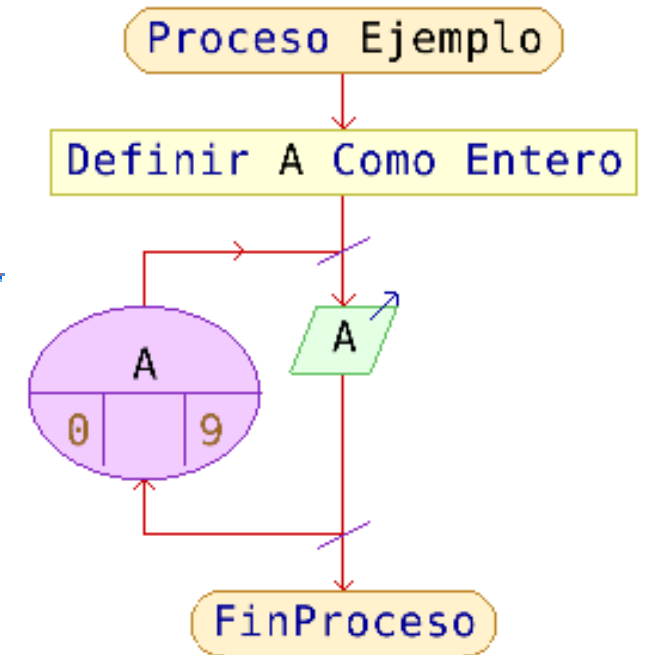
FinProceso



*** Ejecución Iniciada. ***

0
1
2
3
4
5
6
7
8
9

*** Ejecución Finalizada. ***



Operaciones: estructuras de Control

Pseudocódigo

```
Proceso Ejemplo
  Definir A Como Entero;
  Para variable_numerica<-valor_inicial Hasta valor_final Con Paso paso Hacer
    .....
    secuencia_de_acciones
  FinPara
FinProceso
```

Ayuda Rápida

PARA

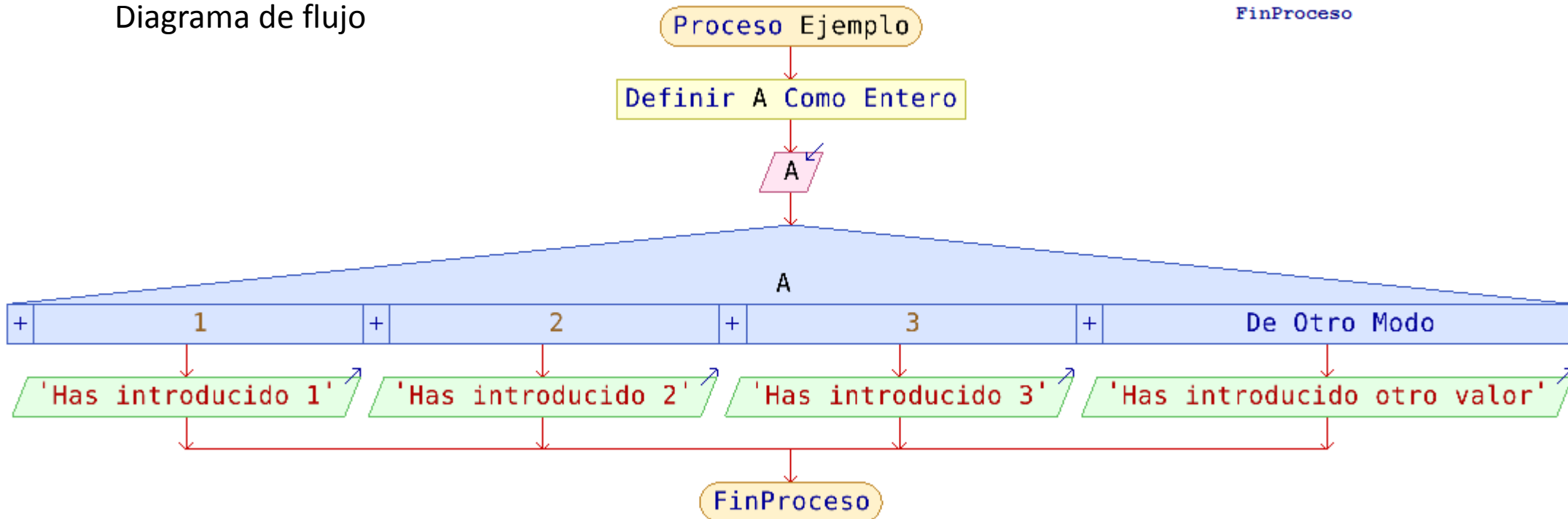
- {variable_numerica}: complete aquí el identificador de la variable que utilizará para iterar.
- {valor_inicial}: introduzca aquí el valor de {variable_numerica} desde el cual se comenzará a iterar.
- {valor_final}: introduzca aquí el valor final de {variable_numerica} hasta el cual se continuará iterando.
- {paso}: introduzca aquí el incremento que se debe realizar a {variable_numerica} en cada iteración.
- {secuencia_de_acciones}: complete aquí la lista de instrucciones que desea repetir en cada iteración.

Proceso Ejemplo

```
Definir A Como Entero;
Para A<-0 Hasta 10 Con Paso 2 Hacer
  .....
  Escribir A;
FinPara
FinProceso
```

Operaciones: estructuras de Control

Diagrama de flujo



Proceso Ejemplo

Definir A Como Entero;

Leer A;

Segun A Hacer

1:

Escribir 'Has introducido 1';

2:

Escribir 'Has introducido 2';

3:

Escribir 'Has introducido 3';

De Otro Modo:

Escribir 'Has introducido otro valor';

FinSegun

FinProceso

INSTALACIÓN PSEINT

<http://pseint.sourceforge.net/>



Descargar Paquete para GNU/Linux 64bits (tgz - 7.5MB)

Descargar Paquete para GNU/Linux 32bits (tgz - 7.3MB)



Descargar Instalador para Microsoft Windows (exe - 7.4MB)



Descargar Paquete para Mac OS i686 (dmg - 25MB)

Descargar Paquete para Mac OS PowerPC (dmg - 25MB)



Descargar Código Fuente (tgz - 1.4MB)

