Pseudocódigo con PsInt

El pseudocódigo es una forma para aprender a programar en castellano, ya que los lenguajes de programación son en inglés.:

- Sintaxis sencilla
- Manejo de las estructuras básicas de control
- Pocos de datos básicos: numérico (Entero o Real), carácter /cadenas de caracteres y lógico (verdadero-falso).
- Estructuras de datos: arreays

Todo algoritmo en pseudocógido tiene la siguiente estructura general:

```
Algoritmo Titulo
     Declaracion de variables
     accion 1;
     accion 2;
                .
                .
                .
          accion n;
FinAlgoritmo
```

Comienza con la palabra clave *Proceso o Algoritmo* seguida del nombre del programa, luego le sigue una secuencia de instrucciones y finaliza con la palabra *FinProceso*. Una secuencia de instrucciones es una lista de una o más instrucciones, cada una terminada en punto y coma.

Las acciones incluyen operaciones de entrada y salida, asignaciones de variables, condicionales si-entonces o de selección múltiple y/o lazos mientras, repetir o para.

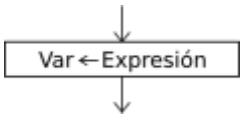
Tipos de datos

- **Entero :** números enteros, sin decimales. Con signo.
- **Real :** números con decimales.
- **Carácter, cadena:** para texto, caracteres.
- **Lógico:** solo dos posibles valores: *Verdadero / Falso.*

Ejemplo de cadenas:

Algoritmo Cadenas
Definir cad como Cadena
Cad ← "hola" /Podemos asignarle un valor inicial a la cadena
Escribir "Introduce tu nombre"
Leer cad
Escribir "Tu nombre tiene ", Longitud(cadena), "caracteres"
Fin Algoritmo

Sentencia de Asignación



La instrucción de asignación permite almacenar un valor en una variable.

```
<variable> <- <expresión> ;
```

Al ejecutarse la asignación, primero se evalúa la expresión de la derecha y luego se asigna el resultado a la variable de la izquierda. El tipo de la variable y el de la expresión deben coincidir.

Entradas de teclado

La instrucción Leer permite ingresar información desde el ambiente.

```
Leer <variable1> , <variable2> , ... ,
<variableN> ;
```

Esta instrucción lee N valores desde el ambiente (en este caso el teclado) y los asigna a las N variables mencionadas. Pueden incluirse una o más variables, por lo tanto el comando leerá uno o más valores.

Salidas por pantalla

La instrucción Escribir permite mostrar valores al ambiente.

```
Escribir <expr1> , <expr2> , ... , <exprN> ;
```

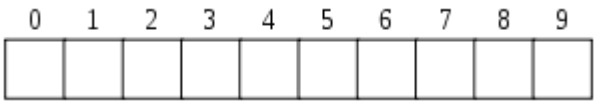
Esta instrucción imprime al ambiente (en este caso en la pantalla) los valores obtenidos de evaluar N expresiones. Dado que puede incluir una o más expresiones, mostrará uno o más valores.

Arrays o listas de valores

En programación se denomina **matriz**, **vector** (de una sola dimensión) o **formación** (en inglés ***array***)[1] a una zona de almacenamiento contiguo que contiene una serie de elementos del mismo tipo, los elementos de la matriz.[2] Desde el punto de vista lógico una matriz se puede ver como un conjunto de elementos ordenados en fila (o filas y columnas si tuviera dos dimensiones).

En principio, se puede considerar que todas las matrices son de una dimensión, la dimensión principal, pero los elementos de dicha fila pueden ser a su vez matrices (un proceso que puede ser recursivo), lo

que nos permite hablar de la existencia de matrices multidimensionales, aunque las más fáciles de imaginar son los de una, dos y tres dimensiones.



Arrays (Dimensionamiento)

La instrucción Dimension permite definir un array, indicando sus dimensiones.

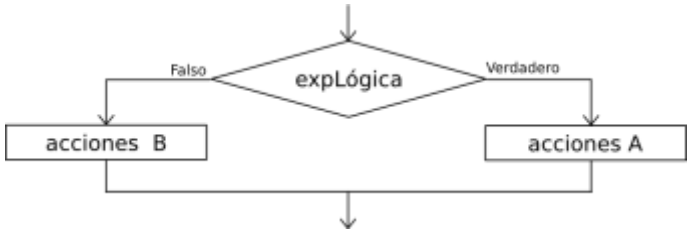
```
Dimesion <identificador> (<max1>,...,<maxN>);
```

Esta instrucción define un array con el nombre indicado en <indentificador> y N dimensiones. Los N parámetros indican la cantidad de dimensiones y el valor máximo de cada una de ellas. La cantidad de dimensiones puede ser una o más, y la máxima cantidad de elementos debe ser una expresión numérica positiva.
Se pueden definir más de un arreglo en una misma instrucción, separándolos con una coma (,).

```
Dimension <ident1> (<max11>,...,<max1N>),..., <identM>(<maxM1>,...,<maxMN>)
```

Es importante notar que es necesario definir un arreglo antes de utilizarlo.

Condicional Si-Entonces



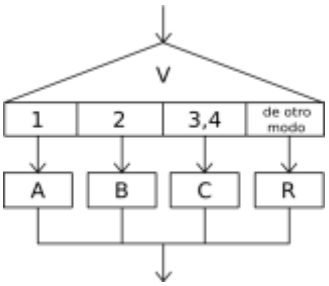
La secuencia de instrucciones ejecutadas por la instrucción Si-Entonces-Sino depende del valor de una condición lógica.

```
Si <condición>
      Entonces
           <instrucciones>
      Sino
            <instrucciones>
FinSi
```

Al ejecutarse esta instrucción, se evalúa la condición y se ejecutan las instrucciones que correspondan: las instrucciones que le siguen al *Entonces* si la condición es verdadera, o las instrucciones que le siguen al *Sino* si la condición es falsa. La condición debe ser una expresión lógica, que al ser evaluada retorna *Verdadero* o *Falso*.
La cláusula *Entonces* debe aparecer siempre, pero la cláusla *Sino* puede no estar. En ese caso, si la

condición es falsa no se ejecuta ninguna instrucción y la ejecución del programa continúa con la instrucción siguiente.

Selección Multiple

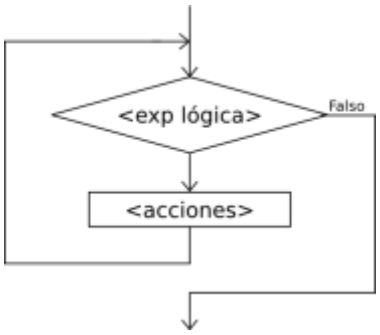


La secuencia de instrucciones ejecutada por una instrucción *Segun* depende del valor de una variable numérica.

```
Segun <variable> Hacer
      <número1>: <instrucciones>
      <número2>,<número3>: <instrucciones>
      <...>
      De Otro Modo: <instrucciones>
FinSegun
```

Esta instrucción permite ejecutar opcionalmente varias acciones posibles, dependiendo del valor almacenado en una variable de tipo numérico. Al ejecutarse, se evalúa el contenido de la variable y se ejecuta la secuencia de instrucciones asociada con dicho valor.
Cada opción está formada por uno o más números separados por comas, dos puntos y una secuencia de instrucciones. Si una opción incluye varios números, la secuencia de instrucciones asociada se debe ejecutar cuando el valor de la variable es uno de esos números.
Opcionalmente, se puede agregar una opción final, denominada *De Otro Modo*, cuya secuencia de instrucciones asociada se ejecutará sólo si el valor almacenado en la variable no coincide con ninguna de las opciones anteriores.

Bucles (o lazos) Mientras



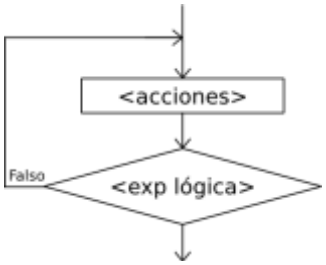
La instrucción *Mientras* **ejecuta** una secuencia de instrucciones **mientras una condición sea verdadera.**

```
Mientras <condición> Hacer
                    <instrucciones>
```

```
FinMientras
```

Al ejecutarse esta instrucción, la condición es evaluada. Si la condición resulta verdadera, se ejecuta una vez la secuencia de instrucciones que forman el cuerpo del ciclo. Al finalizar la ejecución del cuerpo del ciclo se vuelve a evaluar la condición y, si es verdadera, la ejecución se repite. Estos pasos se repiten mientras la condición sea verdadera.
Note que las instrucciones del cuerpo del ciclo pueden no ejecutarse nunca, si al evaluar por primera vez la condición resulta ser falsa.
Si la condición siempre es verdadera, al ejecutar esta instrucción se produce un ciclo infinito. A fin de evitarlo, las instrucciones del cuerpo del ciclo deben contener alguna instrucción que modifique la o las variables involucradas en la condición, de modo que ésta sea falsificada en algún momento y así finalice la ejecución del ciclo.

Bucles “Repetir Hasta”

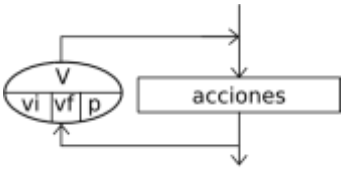


La instrucción *Repetir-Hasta Que* **ejecuta** una secuencia de instrucciones **hasta que la condición sea verdadera.**

```
Repetir
      <instrucciones>
Hasta Que <condición>
```

Al ejecutarse esta instrucción, la secuencia de instrucciones que forma el cuerpo del ciclo se ejecuta una vez y luego se evalúa la condición. Si la condición es falsa, el cuerpo del ciclo se ejecuta nuevamente y se vuelve a evaluar la condición. Esto se repite hasta que la condición sea verdadera.
Note que, dado que la condición se evalúa al final, las instrucciones del cuerpo del ciclo serán ejecutadas al menos una vez.
Además, a fin de evitar ciclos infinitos, el cuerpo del ciclo debe contener alguna instrucción que modifique la o las variables involucradas en la condición de modo que en algún momento la condición sea verdadera y se finalice la ejecución del ciclo.

Bucle Para



La instrucción *Para* **ejecuta** una secuencia de instrucciones **un número determinado de veces**.

```
Para <variable> <- <inicial> Hasta <final> ( Con Paso <paso> ) Hacer
      <instrucciones>
FinPara
```

Al ingresar al bloque, la variable <variable> recibe el valor <inicial> y se ejecuta la secuencia de instrucciones que forma el cuerpo del ciclo. Luego se incrementa la variable <variable> en <paso> unidades y se evalúa si el valor almacenado en <variable> superó al valor <final>. Si esto es falso se repite hasta que <variable> supere a <final>. Si se omite la cláusula *Con Paso* <paso>, la variable <variable> se incrementará en 1.

Operadores y Funciones

Este pseudolenguaje dispone de un conjunto básico de operadores y funciones que pueden ser utilizados para la construcción de expresiones más o menos complejas.

Las siguientes tablas exhiben la totalidad de los operadores de este lenguaje reducido:

Operador	*Significado*	*Ejemplo*
Relacionales		
>	Mayor que	3>2
<	Menor que	'ABC'<'abc'
=	Igual que	4=3
<=	Menor o igual que	'a'<='b'
>=	Mayor o igual que	4>=5
Logicos		
& ó Y	Conjunción (y).	(7>4) & (2=1) //falso
| ó O	Disyunción (o).	(1=1 | 2=1) //verdadero
~ ó NO	Negación (no).	~(2<5) //falso
Algebraicos		
+	Suma	total <- cant1 + cant2
-	Resta	stock <- disp - venta
*	Multiplicación	area <- base * altura
/	División	porc <- 100 * parte / total
^	Potenciación	sup <- 3.41 * radio ^ 2
% ó MOD	Módulo (resto de la división entera)	resto <- num MOD div

La jerarquía de los operadores matemáticos es igual a la del álgebra, aunque puede alterarse mediante el uso de paréntesis.

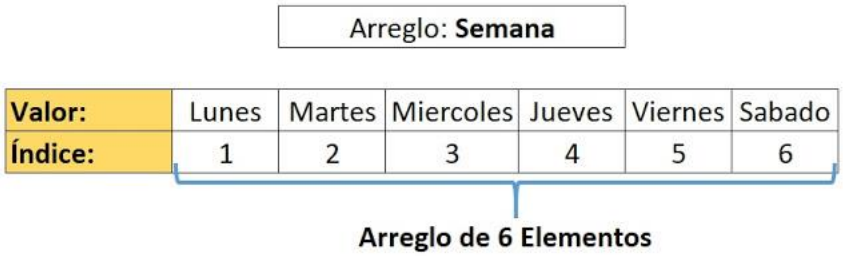
A continuación se listan las funciones integradas disponibles:

Función	*Significado*
RC(X)	Raíz Cuadrada de X
ABS(X)	Valor Absoluto de X
LN(X)	Logaritmo Natural de X
EXP(X)	Función Exponencial de X
SEN(X)	Seno de X
COS(X)	Coseno de X
ATAN(X)	Arcotangente de X
TRUNC(X)	Parte entera de X
REDON(X)	Entero más cercano a X
AZAR(X)	Entero aleatorio entre 0 y X-1

Arrays

Los arrays son estructuras que almacenan datos de un mismo tipo; así podemos imaginar un arreglo como un conjunto de casilleros enumerados "Siendo el primer casillero el 1 (Índice)". Se accede a cada elemento por un índice entero.

Estructura de un array: Dimension Semana[7]



Cómo definir o declarar un array en PSeInt:

Dimension *nombre_array [tamaño_array]*

Por ejemplo: array para almacenar 5 números:

Dimension numeros[5]

Cómo insertar valores en el array

Llenar un arreglo es muy sencillo, solo debemos indicar el valor que irá en determinada posición, siguiendo con el arreglo de los 5 números sería:

```
Algoritmo arreglos
    Dimension numeros[5]
    numeros[1] <- 2
    numeros[2] <- 4
    numeros[3] <- 9
    numeros[4] <- 15
    numeros[5] <- 6
FinAlgoritmo
```

Como podemos observar indicamos el lugar en el que almacenamos cada número.

Una forma rápida de leer los datos que están guardados en las posiciones de un array es haciendo referencia a su índice.

```
Algoritmo arreglos
    Dimension numeros[5]
    numeros[1] <- 2
    numeros[2] <- 4
    numeros[3] <- 9
    numeros[4] <- 15
    numeros[5] <- 6
    Escribir "Índice 1: ",numeros[1]
    Escribir "Índice 2: ",numeros[2]
    Escribir "Índice 3: ",numeros[3]
    Escribir "Índice 4: ",numeros[4]
    Escribir "Índice 5: ",numeros[5]
FinAlgoritmo
```

El código anterior imprimiría en pantalla:

```
Índice 1: 2
Índice 2: 4
Índice 3: 9
Índice 4: 15
Índice 5: 6
```

De la forma anterior podemos obtener valores de un array, sin embargo estas estructuras comúnmente son utilizadas para almacenar muchos datos por lo que se torna complicado saber con exactitud el índice deseados; es por ello que al manejar arreglos, debemos remitirnos al uso de ciclos repetitivos; es necesario el uso de ciclos dado que un arreglo contiene múltiples valores, así para obtener cada valor almacenado debemos "*Recorrer*" el arreglo.

```
Algoritmo arreglos
    Dimension numeros[5]
    numeros[1] <- 2
    numeros[2] <- 4
    numeros[3] <- 9
    numeros[4] <- 15
    numeros[5] <- 6
    Para i<-1 Hasta 5 Con Paso 1 Hacer
        Escribir "Índice ",i, " contiene: ",numeros[i]
    Fin Para
FinAlgoritmo
```

Como observamos en el anterior bloque de código, recorremos todas las posiciones del array, mediante la variable "*i*", obtenemos el valor en cada iteración del ciclo, imprimiendo:

Algunas Observaciones

- Se pueden introducir comentarios luego de una instrucción, o en líneas separadas, mediante el uso de la doble barra (*//*). Todo lo que precede a *//*, hasta el fin de la línea, no será tomado en cuenta al interpretar el algoritmo.

- Notese que no puede haber instrucciones fuera del programa, aunque si comentarios.

- Las estructuras no secuenciales pueden anidarse. Es decir, pueden contener otras adentro, pero la estructura contenida debe comenzar y finalizar dentro de la contenedora.

- Los identificadores, o nombres de variables, deben constar sólo de letras y números, comenzando siempre con una letra, y no pueden ser palabras reservadas (como para, mientras, y, no, etc...)

- Las constantes de tipo carácter se escriben entre comillas (").

- En las constantes numéricas, el punto (.) es el separador decimal.

- Las constantes lógicas son *Verdadero* y *Falso*.

SUBRUTINAS Y FUNCIONES

En muchos casos, nos encontraremos con tareas que tenemos que repetir varias veces en distintos puntos de nuestro programa. Si tecleamos varias veces el mismo fragmento de programa no sólo tardaremos más en escribir: además el programa final resultará menos legible, será más también será más fácil que cometamos algún error alguna de las veces que volvemos a teclear el fragmento repetitivo, o que decidamos hacer una modificación y olvidemos hacerla en alguno de los fragmentos. Por eso, conviene evitar que nuestro programa contenga código repetitivo. Una de las formas de evitarlo es usar "subrutinas", una posibilidad que la mayoría de lenguajes de programación permiten, y que en ocasiones recibe el nombre de "procedimientos" o de "funciones" (existe algún matiz que hace que esas palabras no sean realmente sinónimas y que comentaremos más adelante).

Una subrutina es una porción de código que realiza una tarea específica y relativamente independiente del resto del código.

PseInt permite definir "subrutinas" (o "funciones") dentro del pseudocódigo, desde la versión del 10 de octubre de 2012. En su caso, se llaman "subprocesos". Veamos algunos ejemplos de su uso:

```
Funcion Saludar

     Escribir "Hola mundo!"

FinFuncion
```

Función que recibe un argumento por valor, y devuelve su doble

```
Funcion res <- CalcularDoble(num)

     res <- num*2 // retorna el doble

FinFuncion
```

Función que recibe un argumento por referencia, y lo modifica

```
Funcion Triplicar(num por referencia)

     num <- num*3 // modifica la variable duplicando su valor

FinFuncion
```

Ejemplos de subprocesos:

```
SubProceso Saludar
    Escribir "Hola mundo!"
FinSubProceso

// funcion que recibe un argumento por valor, y devuelve su doble
SubProceso res <- CalcularDoble(num)
    res <- num*2 // retorna el doble
FinSubProceso

// funcion que recibe un argumento por referencia, y lo modifica
SubProceso Triplicar(num por referencia)
    num <- num*3 // modifica la variable duplicando su valor
FinSubProceso

// proceso principal, que invoca a las funciones antes declaradas
Proceso PruebaFunciones


    Escribir "Llamada a la funcion Saludar:"
    Saludar // como no recibe argumentos pueden omitirse los paréntesis
vacios

    Escribir "Ingrese un valor numérico para x:"
    Leer x

    Escribir "Llamada a la función CalcularDoble (pasaje por valor)"
    Escribir "El doble de ",x," es ", CalcularDoble(x)
    Escribir "El valor original de x es ",x

    Escribir "Llamada a la función Triplicar (pasaje por referencia)"
    Triplicar(x)
    Escribir "El nuevo valor de x es ", x

FinProceso
```