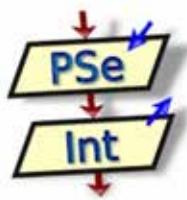


PROGRAMACIÓN DESDE CERO

INTRODUCCIÓN A LA PROGRAMACIÓN CON PSEINT



Universidad de
LA PUNTA

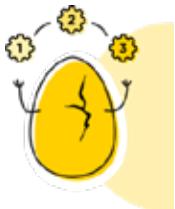


GOBIERNO DE
SAN LUIS



¡NUESTRA PRIMERA GUÍA!

Esta es tu primera guía, en la que vamos a empezar a ver algunos conceptos teóricos del mundo de la programación y el desarrollo web. Pero antes, queremos explicarte cómo debes leer las guías y qué encontrarás de ahora en adelante.



Cuando veas el indicador AMARILLO, estaremos explicando metodología de trabajo. Cómo deberás trabajar con tu equipo.



El indicador NARANJA te guiará respecto al contenido que estudiarás en esa guía. Te permitirá llevar un control de los conocimientos que debas adquirir y a hacer un seguimiento del mismo. No sólo en los aspectos técnicos, sino también en la implementación de la metodología. Te recomendamos que te tomes un tiempo para analizar si comprendiste correctamente la teoría, los ejemplos y pudiste aplicar lo aprendido en los ejercicios.



El indicador AZUL te mostrará ejemplos gráficos de la teoría y prácticos. Esto te permitirá esclarecer conceptos y llevar a la práctica los mismos.



El indicador MORADO te invitará a poner en práctica el concepto sobre el cual estabas leyendo. Suelen ser ejercicios muy breves en donde aplicarás sólo un concepto, o aquellos que ya hayas aprendido, más el que estás incorporando.

Encontrarás otro apartado llamado "MANOS A LA OBRA – CORRECCIÓN DE ERRORES" en dónde deberás copiar y pegar un ejercicio en tu IDE, chequear los errores en la consola y corregirlos de tal manera que, al ejecutar el programa, la consola se vea igual a la captura del ejercicio.



El indicador VERDE te proveerá tips, consejos y recomendaciones.



Objetivos de la Guía

En esta guía aprenderemos a:

- Definir todos los tipos de variables y nombrarlas correctamente.
- Asignarles valor a las variables.
- Utilizar métodos de escritura para mostrar mensajes por pantalla.
- Utilizar métodos de lectura para ingresar valores por teclado y alojarlo en las variables.
- Operar con las variables.

¿QUÉ ES LA PROGRAMACIÓN?

La programación es el acto de programar, es decir, **organizar una secuencia de pasos ordenados a seguir para hacer cierta cosa**. Este término puede utilizarse en muchos contextos. Por ejemplo, programar una salida con amigos, organizar unas vacaciones, etc.

En informática el término **programación** se refiere a la acción de crear programas y **programar** es la serie de instrucciones, que le vamos a dar a nuestra máquina, para lograr lo que nuestro programa necesite para funcionar.

Las partes que componen a nuestro programa son **el lenguaje de programación y los algoritmos**.



¡MANOS A LA OBRA!

EJERCICIO ALARMA

Ahora que sabemos que es programar, antes de ver en profundidad lenguajes de programación y algoritmos, vamos a programar algo muy sencillo.

Vamos a programar una alarma en el celular. Deberás entrar a su celular y poner una alarma para que suene a la hora en la que debes conectarte a tu clase, lo importante es que vean como al celular le damos una serie de instrucciones para realizar una tarea, que sería, hacer ruido a una hora que nosotros decidimos.

¿QUÉ ES UN LENGUAJE DE PROGRAMACIÓN?

Es un lenguaje formal que, mediante una serie de instrucciones, le permite a un programador escribir un conjunto de órdenes, acciones consecutivas, datos y algoritmos para, de esa forma, **resolver problemas**.

Las instrucciones que sigue la computadora para la creación de programas están escritas en un lenguaje de programación y luego son traducidas a un lenguaje de máquina que puede ser interpretado y ejecutado por el hardware del equipo.

Hay distintos tipos de lenguajes de programación:

- **Lenguaje máquina:** Es el más primitivo de los lenguajes y es una colección de dígitos binarios o bits (0 y 1) que la computadora lee e interpreta y son los únicos idiomas que las computadoras entienden. Ejemplo: **10110000 01100001**
- **Lenguajes de alto nivel:** Tienen como objetivo facilitar el trabajo del programador, ya que utilizan unas instrucciones más fáciles de entender.

Además, el lenguaje de alto nivel permite escribir códigos mediante idiomas que conocemos (español, inglés, etc.) y luego, para ser ejecutados, se traduce al lenguaje máquina mediante traductores o compiladores.

¿QUÉ ES UN ALGORITMO?

En el apartado anterior vimos que los lenguajes de programación son nuestro puente para poder comunicarnos con la máquina. Y de esa forma **darle instrucciones claras**, para poder **solucionar los problemas** que puede presentar la creación de un programa.

Estas instrucciones que le vamos a dar a nuestro programa, se conocen **como algoritmos**. Un algoritmo es un **método** para darle instrucciones a nuestro programa y resolver un problema.

Este consiste en la realización de un conjunto de pasos *lógicamente ordenados* tal que, partiendo de la información que le demos, permite obtener ciertos resultados que conforman la solución del problema.

Los algoritmos son **independientes** tanto del lenguaje de programación en que se expresan como de la computadora que los ejecuta. En cada problema el algoritmo se puede expresar en un lenguaje diferente de programación y ejecutarse en una computadora distinta; sin embargo, el algoritmo **será siempre el mismo**. Así, por ejemplo, en una analogía con la vida diaria, una receta de un plato de cocina se puede expresar en español, inglés o francés, pero cualquiera que sea el lenguaje, los pasos para la elaboración del plato se realizan sin importar el idioma del cocinero.

Los **algoritmos** son más importantes que los **lenguajes de programación** o las **computadoras**. Un lenguaje de programación es sólo un medio para expresar un algoritmo y una computadora es solo un procesador para ejecutarlo. Tanto el lenguaje de programación como la computadora son los medios para obtener un fin: conseguir que el algoritmo se ejecute y se efectúe el proceso correspondiente.

CARACTERÍSTICAS FUNDAMENTALES DE LOS ALGORITMOS

- Un algoritmo debe ser **preciso** e indicar el orden de realización de cada paso.
- Un algoritmo debe estar específicamente **definido**. Es decir, si se ejecuta un mismo algoritmo dos veces, con los mismos datos de entrada, se debe obtener el mismo resultado cada vez.
- Un algoritmo debe ser **finito**. Si se sigue un algoritmo, se debe terminar en algún momento; o sea, debe tener un número finito de pasos. Debe tener un inicio y un final.
- Un algoritmo debe ser **correcto**: el resultado del algoritmo debe ser el resultado esperado.
- Un algoritmo es **independiente** tanto **del lenguaje de programación** en el que se expresa **como de la computadora** que lo ejecuta.

El programador debe constantemente resolver problemas de manera algorítmica, lo que significa plantear el problema de forma tal que queden indicados los pasos necesarios para obtener los resultados pedidos, a partir de los datos conocidos. Lo anterior implica que un algoritmo básicamente consta de tres elementos: *Datos de Entrada o Información de entrada, Procesos y la Información de Salida*.



Entrada → Procesos → Salida

Estructura de un Programa: Datos de entrada, proceso y Salida.

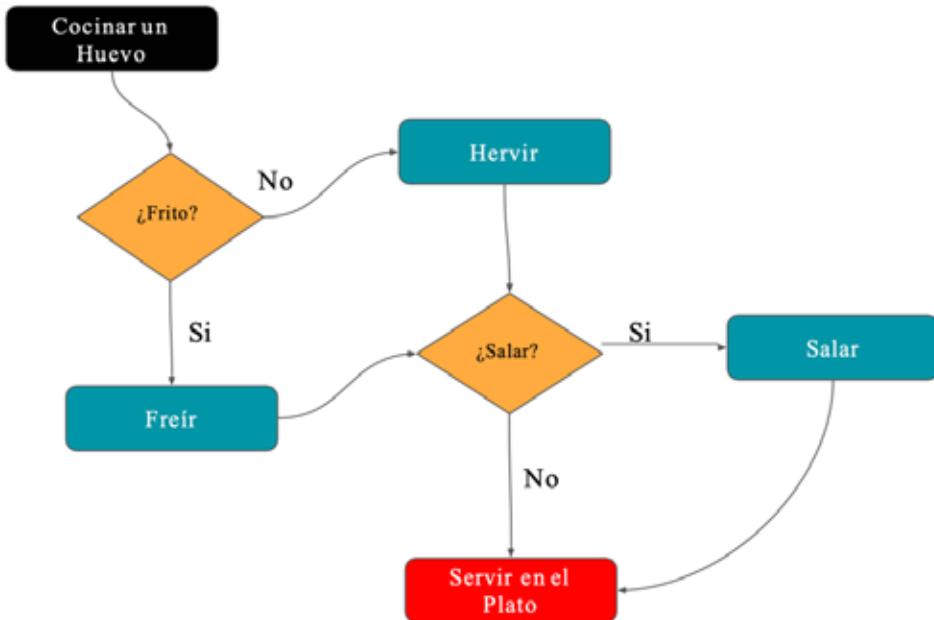
¿CÓMO REPRESENTAR UN ALGORITMO?

La mejor manera de representar un algoritmo es a través de un diagrama de flujo.

Un Diagrama de Flujo representa la esquematización gráfica de un algoritmo, el cual muestra gráficamente los pasos o procesos a seguir para alcanzar la solución de un problema.

Se basan en la utilización de diversos símbolos para representar operaciones específicas, es decir, es la representación gráfica de las distintas operaciones que se tienen que realizar para resolver un problema, con indicación expresa el orden lógico en que deben realizarse.

Vamos a ver cómo sería un diagrama de flujo, de los pasos para cocinar un huevo:



Como podemos ver, nos muestra paso a paso cómo cocinar un huevo y cada opción que tenemos a la hora de cocinar un huevo.

¿QUÉ ES EL PSEUDOCÓDIGO Y POR QUÉ VAMOS A UTILIZARLO?

El lenguaje de programación que utilizaremos en **esta parte del curso**, para representar nuestros algoritmos es el pseudocódigo.

El **pseudocódigo** es una herramienta de programación en la que las instrucciones se escriben en palabras similares al inglés o español, que facilitan tanto la escritura como la lectura de programas. En esencia, el pseudocódigo se puede definir como un lenguaje de especificaciones de algoritmos. El uso de tal lenguaje hace el paso de codificación final (esto es, la traducción a un lenguaje de programación) relativamente fácil. El pseudocódigo se considera un primer borrador, dado que tiene que traducirse posteriormente a un lenguaje de programación.

PROGRAMA

En los apartados anteriores, explicamos qué es la programación y que está compuesta de dos partes, los lenguajes de programación y los algoritmos. Pero, ¿dónde se van a ver reflejados estos dos conceptos? Estos se van a ver reflejados en nuestro **programa**.

Un programa no es más que una **serie de algoritmos escritos en algún lenguaje de programación de computadoras**. Un programa es, por lo tanto, un conjunto de instrucciones —órdenes dadas a la computadora— que producirán la ejecución de una determinada tarea. En esencia, un programa es un medio para conseguir un fin. El fin será probablemente definido como la información necesaria para solucionar un problema.

DISEÑO DEL PROGRAMA

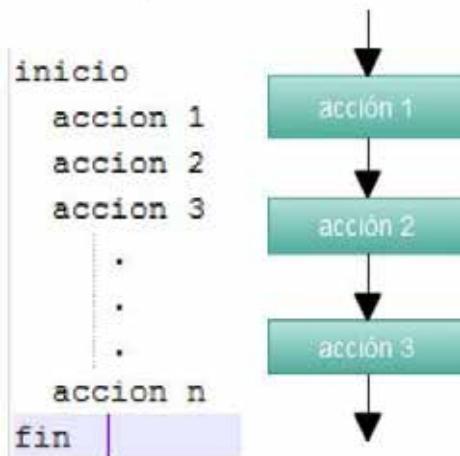
Se puede utilizar algunas de las herramientas de representación de algoritmos, también conocidos como lenguajes de programación, para definir la secuencia de pasos que se deben llevar a cabo para conseguir el resultado que necesitamos.

ESPECIFICACIONES DE UN PROGRAMA

Tras la decisión de desarrollar un programa, el programador debe establecer el conjunto de especificaciones que debe contener el programa: entrada, salida y algoritmos de resolución, que incluirán las técnicas para obtener las salidas a partir de las entradas.

Un programa puede ser lineal (secuencial) o no lineal. Un programa es lineal si las instrucciones (acciones) se ejecutan secuencialmente como los ejercicios propuestos en esta guía, es decir, sin bifurcaciones, decisión ni comparaciones.

Pseudocódigo



Estructura de un programa secuencial

CODIFICACIÓN

Una vez que tenemos las especificaciones de un programa pasaremos a la codificación del programa. La codificación es la operación de escribir la solución del problema (de acuerdo a la lógica del pseudocódigo), en una serie de instrucciones detalladas, en un código reconocible por la computadora. La serie de instrucciones detalladas se conoce como código fuente, el cual se escribe en un lenguaje de programación o lenguaje de alto nivel.



Leímos muchos conceptos nuevos hasta aquí, es un buen momento para acudir a tu equipo si tienes alguna duda.

¿CÓMO DEBEN ESCRIBIRSE LOS ALGORITMOS/PROGRAMAS?

Ya sabemos qué es un programa, el diseño de un programa, las especificaciones de un programa y su codificación. Ahora vamos a ver cómo es la escritura de estos algoritmos / programas. Un algoritmo consta de dos componentes: una cabecera de programa y un bloque algoritmo. La cabecera de programa es una acción simple que comienza con la palabra algoritmo. Esta palabra estará seguida por el nombre asignado al programa completo.

El bloque algoritmo es el resto del programa y consta de dos componentes o secciones: las *acciones de declaración* y las *acciones ejecutables*.

Las declaraciones definen o declaran las variables que tengan nombres. Las acciones ejecutables son las acciones que posteriormente deberá realizar la computación cuando el algoritmo convertido en programa se ejecute.

algoritmo
cabecera del programa
sección de declaración
sección de acciones

CABECERA DEL PROGRAMA

Todos los algoritmos y programas deben comenzar con una cabecera en la que se exprese el identificador o nombre correspondiente con la palabra reservada que señale el lenguaje. En PSeInt, la palabra reservada es Algoritmo.

Algoritmo sin_titulo

<Acciones>

FinAlgoritmo

Donde la palabra sin título debe ser reemplazada por el nombre del algoritmo. Esto se vería así en PseInt.



¿NECESITAS UN EJEMPLO?

```
1 Algoritmo sin_titulo
2
3
4 FinAlgoritmo
5
```

¿QUÉ ELEMENTOS POSEE UN PROGRAMA?

Los elementos de un programa, son básicamente, los componentes que conforman las instrucciones previamente mencionadas, para crear nuestro programa y resolver sus problemas. Estos elementos siempre estarán dentro de un algoritmo.

Los elementos de un programa son: **identificadores, variables, constantes, operadores, palabras reservadas**.



¿No recuerdas cómo identificar al facilitador de tu equipo?
Ve a tu Aula Virtual. En la barra lateral podrás ver a tu equipo de trabajo del día. Quien tiene una insignia es el/la facilitador/a del equipo.

IDENTIFICADORES

Un identificador es un conjunto de caracteres alfanuméricos de cualquier longitud que sirve para identificar las entidades del programa (nombre del programa, nombres de variables, constantes, subprogramas, etc.). En PseInt los identificadores deben constar sólo de **letras, números y/o guión_bajo(_)**, comenzando siempre con una letra y **se suelen escribir siempre en minúsculas**. Estos tampoco pueden contar de tildes, ni de la letra Ñ, ya que generaría errores.

Otra cosa que es súper importante a la hora de pensar identificadores, es **poner nombres claros**, por ejemplo, si queremos tener una frase, que el identificador sea **frase** o si queremos una suma, le pondremos **suma**.



¿Y si necesitamos un identificador de más de una palabra?

Ahora, si queremos poner un identificador que sea de **más de una palabra**, usamos algo llamado **camelCase**. Consiste en poner la letra de la segunda palabra en mayúsculas, por ejemplo, queremos que el identificador refleje que suma números, le pondríamos **sumaNumeros**, la segunda palabra arranca con la N en mayúsculas. De esta manera, mantenemos la regla de escribir siempre en minúsculas.

Esto no es solo para la segunda palabra, si queremos poner una tercera o una cuarta palabra, etc. haríamos lo mismo. Por ejemplo, **sumaNumerosEnteros**, para un identificador más claro.

VARIABLES Y CONSTANTES

Los programas de computadora necesitan **información** para la resolución de problemas. Esta información puede ser un número, un nombre, etc. Para nosotros poder guardar esta información en algún lugar y que no esté “suelta”, para no perderla o poder acceder a ella cuando lo necesitemos es crucial. Para solucionar esto, vamos a guardar la información en algo llamado, **variables y constantes**. Las variables y constantes vendrían a ser como pequeñas cajas, que guardan algo en su interior, en este caso información. Estas, van a contar como previamente habíamos mencionado, con un identificador, un nombre que facilitará distinguir unas de otras y nos ayudará a saber que variable o constante es la que contiene la información que necesitamos.

Dentro de toda la información que vamos a manejar, a veces, necesitaremos información que no cambie. Tales valores son constantes. De igual forma, existen otros valores que necesitaremos que cambien durante la ejecución del programa; esas van a ser nuestras **variables**.

Una variable es un objeto o tipo de datos cuyo valor puede cambiar durante el desarrollo del algoritmo o ejecución del programa. Dependiendo del lenguaje, hay diferentes tipos de variables, tales como enteras, reales, carácter, lógicas y de cadena. Una variable que es de un cierto tipo puede tomar únicamente valores de ese tipo. Una variable de carácter, por ejemplo, puede tomar como valor sólo caracteres, mientras que una variable entera puede tomar sólo valores enteros.

Ejemplo: una variable que guardará el resultado de un cálculo, este valor puede cambiar, en alguna parte de nuestro programa.

Una constante es un dato, que al igual que la variable, puede ser de diferentes tipos como enteras, reales, carácter, lógicas y de cadena. Estas, también guardan solo valores de ese tipo, pero, permanecen sin cambios durante todo el desarrollo del algoritmo o durante la ejecución del programa. **Ejemplo:** el valor de Pi π

TIPOS DE DATOS EN PSEINT

Las variables y constantes como previamente habíamos mencionado, van a guardar información dependiendo del **tipo de dato** que le digamos que guarde esa variable. Por ejemplo, si digo que mi variable va a guardar números enteros, significa que el tipo de dato de esa variable es entero.

Los tipos de datos que podemos usar son:

- Entero: solo números enteros.
- Real: números con cifras decimales. Para separar decimales se utiliza el punto. Ejemplo: **3.14**
- Carácter: cuando queremos guardar un carácter. Los Caracteres se encierran entre comillas simples. un carácter (unidimensional): '**a**', '**A**'.
- Lógico: cuando necesitamos guardar una expresión lógica (verdadero o falso). Es el equivalente de encendido/apagado, 0/1. Estas expresiones las usaremos mucho a medida que avance el curso para poner condiciones.
- Cadena: cuando queremos guardar cadenas de caracteres. Las Cadenas se encierran entre comillas dobles. una cadena (multidimensional): "**esto es una cadena**", "**hola mundo**"

Notas:

- Cadena y Carácter son términos equivalentes, no genera error que las escribamos indistintamente.
- El plural de Carácter es Caracteres o Cadena.

¿CÓMO CREO UNA VARIABLE?

A la hora de crear nuestra variable, vamos a tener que darle un **identificador, por lo que usaremos las reglas vistas en el apartado de identificadores**, y el tipo de dato que necesitamos que guarde. Para esto vamos a utilizar la palabra reservada **Definir**. La instrucción definir permite explicitar el tipo de una o más variables.

Después de la palabra **Definir**, va el nombre de la variable y por último el tipo de dato de la variable. Normalmente los identificadores de las variables y de las constantes con nombre deben ser declaradas en los programas antes de ser utilizadas. Entonces, la sintaxis de la declaración de una variable suele ser:

Definir <nombre_variable> como <tipo_de_dato>

Si queremos declarar una variable de tipo entero se escribe:

Definir varNumero Como Entero

varNumero se convierte en una variable de tipo entero.



NECESITAS UN EJEMPLO?

Definir num Como Entero



¡MANOS A LA OBRA!

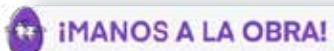
EJERCICIO CREAR VARIABLE

Vamos a poner en práctica lo que acabamos de ver, vamos a crear en PseInt, una variable de cada tipo de dato posible.



Palabras Reservadas

Palabras que dentro del lenguaje significan la ejecución de una instrucción determinada, por lo que no pueden ser utilizadas con otro fin. En Pseudocódigo, las palabras reservas aparecen de color azul. Por ejemplo, la palabra **Algoritmo** y **FinAlgoritmo**.



DETECCIÓN DE ERRORES

Copia y pega el código que está a continuación, ahí vas a ver que el código tiene mal escritas las palabras reservadas, tu tarea es arreglar el código.

Algoritmo Definicion_Variables

Definir algoritmo Como Logico

Definir numero Como Entero

Definir cadena Como Caracter

FinAlgoritmo



Repasemos lo que has aprendido hasta aquí

- Definir variables de tipo lógico, entero, real y cadena.
- Nombrar correctamente las variables sin utilizar palabras reservadas.

TIPOS DE INSTRUCCIONES

Las instrucciones —acciones— básicas que se pueden implementar de modo general en un algoritmo y que esencialmente soportan todos los lenguajes son las siguientes:

INSTRUCCIONES DE INICIO/FIN

Son utilizadas para delimitar bloques de código. Por ejemplo, Algoritmo y FinAlgoritmo.

INSTRUCCIONES DE ESCRITURA O SALIDA

Ayer estuvimos trabajando un poco con esto, ahora si, te explicaremos cómo funcionan estas instrucciones.

Se utilizan para escribir o mostrar mensajes o contenidos de las variables en un dispositivo de salida. La salida puede aparecer en un dispositivo de salida (pantalla, impresora, etc.).

La operación de salida se denomina escritura (escribir). En la escritura de algoritmos las acciones escritura se representa con el siguiente formato:

Escritura o salida

La instrucción Escribir permite mostrar información y valores de variables en la interfaz grafica o ambiente.

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

Esta instrucción imprime en la interfaz grafica o ambiente (en este caso en la pantalla) los valores obtenidos de evaluar N expresiones, el valor de un variable o de un mensaje. Dado que puede incluir una o más expresiones, mostrará uno o más valores.



¿NECESITAS UN EJEMPLO?



```
1 Algoritmo ejemploEscribir *** Ejecución Iniciada. ***
2   Definir num Como Entero Hola mundo
3
4   Escribir "Hola mundo" 4
5   Escribir "2 + 2" 10
6
7   Escribir 2 + 2 *** Ejecución Finalizada. ***
8
9   num = 10
10
11  Escribir num
12
13
14 FinAlgoritmo
```

En este ejemplo de escribir, vemos que nuestro primer escribir muestra un mensaje o cadena, que va entre comillas dobles, después nuestro segundo escribir muestra el resultado de una suma de dos números y nuestro último escribir, muestra el valor de una variable de tipo entero a la que se le asignó un valor previo.

Con el escribir también podemos mostrar variables o valores con un mensaje previo, para esto vamos a concatenar nuestra variable, usando una coma o un espacio en blanco, con un mensaje entre comillas.



¿NECESITAS UN EJEMPLO?

Escribir “Mensaje entre comillas” variable

Escribir “La suma de los números es:” suma



EJERCICIO ESCRIBIR

Escribir un algoritmo en el cual se muestre nuestro nombre completo en la interfaz gráfica de PseInt.



Repasemos lo que has aprendido hasta aquí

- Utilizar instrucciones de escritura para mostrar mensajes por pantalla al ejecutar el programa.

INSTRUCCIONES DE LECTURA

Los cálculos que realizan las computadoras requieren, para ser útiles la entrada de los datos necesarios para ejecutar las operaciones que posteriormente se convertirán en resultados, es decir, salida.

Las operaciones de entrada permiten leer datos de un dispositivo de entrada y asignarlos a determinadas variables.

Esta entrada se conoce como operación de lectura (leer). Los datos de entrada se introducen al procesador mediante dispositivos de entrada (teclado, tarjetas perforadas, unidades de disco, etc.).

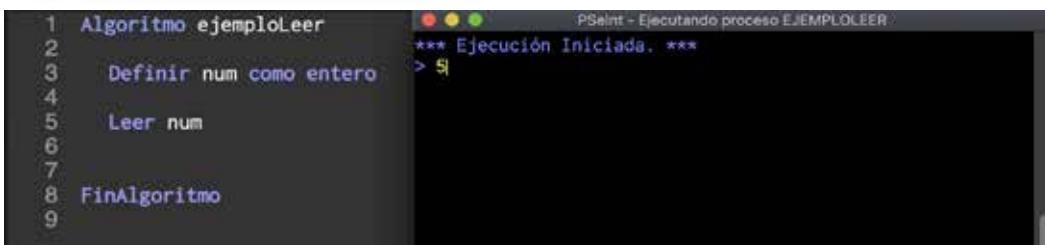
Lectura o entrada

La instrucción Leer permite ingresar información por teclado al usuario a través de la interfaz gráfica o ambiente de Pseint. Que se mostrará al correr nuestro algoritmo

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



¿NECESITAS UN EJEMPLO?



The screenshot shows the PseInt environment. On the left, the algorithm code is displayed:

```
1 Algoritmo ejemploLeer
2
3   Definir num como entero
4
5   Leer num
6
7
8 FinAlgoritmo
9
```

On the right, the execution window shows the output of the program:

```
PSeInt - Ejecutando proceso EJEMPLOLEER
*** Ejecución Iniciada. ***
> 5
```

En este ejemplo definimos una variable de tipo entero llamada **num** y le asignamos un valor a través de la instrucción **Leer**.



EJERCICIO LECTURA Y ESCRITURA

Escribir un algoritmo en el cual el programa nos pida nuestro nombre completo, lo aloje en una variable de tipo Cadena llamada nombreCompleto y después mostrarlo usando la instrucción Escribir.



Repasemos lo que has aprendido hasta aquí

- Utilizar instrucciones de lectura para pedirle al usuario que guarde valores en las variables por pantalla al ejecutar el programa.

¿CÓMO ASIGNAMOS VALORES A LAS VARIABLES?

La instrucción de asignación permite almacenar un valor en una variable (previamente definida). Esta es nuestra manera de guardar información en una variable, para utilizar ese valor en otro momento. Se puede realizar de dos maneras, con el signo igual o una flecha:

<variable> <- <expresión>

<variable> = <expresión>

expresión es igual a una expresión matemática o lógica, a una variable o constante.

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.



¿NECESITAS UN EJEMPLO?

```
1 Algoritmo ejemploAsignacion
2
3   Definir num Como Entero
4
5   num = 4
6
7 FinAlgoritmo
8
```

En este ejemplo estamos definiendo una variable como entero y después asignándole un valor, en este caso el número 4.



EJERCICIO DEFINIR

Escribe un algoritmo definiendo la variable nombre como cadena, asigna allí tu información y luego muéstralos por pantalla escribiendo la variable.

OPERADORES

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

OPERADORES ALGEBRAICOS

Los operadores algebraicos o también conocidos como operadores aritméticos. Realizan operaciones aritméticas básicas: suma, resta, multiplicación, división, potenciación y modulo para datos de tipo numérico tanto enteros como reales. Estas operaciones son binarias porque admiten dos operandos.

Operador	Significado	Resultado
Algebraicos		
+	Suma	suma = 2 + 2
-	Resta	resta = 10 - 4
*	Multiplicación	multiplicación = 10 * 2
/	División	división = 9 / 3
^	Potenciación	potencia = 10 ^ 2
% o MOD	Módulo (resto de la división entera)	resto = 4 % 2

Reglas de prioridad:

Las expresiones que tienen dos o más operadores requieren unas reglas matemáticas que permitan determinar el orden de las operaciones, se denominan reglas de prioridad y son:

1. Las operaciones que están encerradas entre paréntesis se evalúan primero. Si existen diferentes paréntesis anidados (interiores unos a otros), las expresiones más internas se evalúan primero.
2. Las operaciones aritméticas dentro de una expresión suelen seguir el siguiente orden de prioridad:
 - operador ()
 - operadores unitarios (potenciación),
 - operadores *, /, % (producto, división, módulo) √ operadores +, - (suma y resta).

En caso de coincidir varios operadores de igual prioridad en una expresión o sub expresión encerrada entre paréntesis, el orden de prioridad en este caso es de *izquierda a derecha*, y a esta propiedad se denomina asociatividad.



¡MANOS A LA OBRA!

EJERCICIO VARIABLES

Define dos variables que guarden números enteros, defina una tercera variable donde aloje el resultado se sumarán ambas variables. Comente su código indicando qué finalidad tiene cada línea.

DETECCIÓN DE ERRORES

¿Puedes corregir esta porción de código para que cumpla el resultado esperado?

Algoritmo Prueba

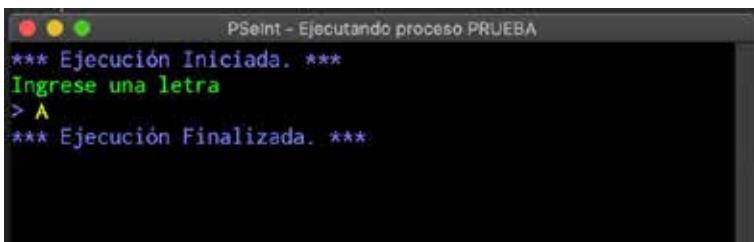
Definir letra Como Entero

Escribir ingrese una letra

letra

FinAlgoritmo

¿CUÁL ES EL RESULTADO A LOGRAR?



```
PSeint - Ejecutando proceso PRUEBA
*** Ejecución Iniciada. ***
Ingresé una letra
> A
*** Ejecución Finalizada. ***
```



Repasemos lo que has aprendido hasta aquí

- Asignar valor a las variables manualmente y operar con ellas.

!

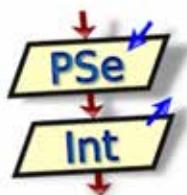
!

!PROGRAMACIÓN DESDE CERO

EJERCICIOS

Introducción a la programación con PSeInt

ENCUENTRO 3



Universidad de
LA PUNTA



!

EJERCICIOS PRÁCTICOS

Vamos a poner en práctica todo lo que hemos visto en esta guía con los siguientes ejercicios.
¡Disfrutemos de nuestro aprendizaje!



Te recordamos que para poder resolver los ejercicios debes haber visto en el encuentro anterior los videos relacionados con este tema. Los podrás encontrar en tu aula virtual o en nuestro canal de YouTube.

Los ejercicios van a tener el siguiente filtro de dificultad:

Dificultad Baja

Dificultad Media

Dificultad Alta

1.! Un colegio desea saber qué porcentaje de niños y qué porcentaje de niñas hay en el curso actual. Diseñar un algoritmo para este propósito. Recuerda que para calcular el porcentaje puedes hacer una regla de 3 simple. El programa debe solicitar al usuario que ingrese la cantidad total de niños, y la cantidad total de niñas que hay en el curso.

2.! Solicitar al usuario que ingrese la base y altura de un rectángulo, y calcular y mostrar por pantalla el área y perímetro del mismo

$$\text{área} = \text{base} * \text{altura}$$

$$\text{perímetro} = 2 * \text{altura} + 2 * \text{base}.$$

3.! Escribir un programa que calcule el volumen de un cilindro. Para ello se deberá solicitar al usuario que ingrese el radio y la altura. Mostrar el resultado por pantalla.

$$\text{volumen} = \pi * \text{radio}^2 * \text{altura}$$

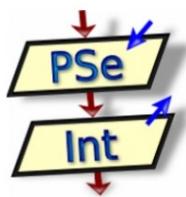
4.! A partir de una conocida cantidad de días que el usuario ingresa a través del teclado, escriba un programa para convertir los días en horas, en minutos y en segundos. Por ejemplo

$$1 \text{ día} = 24 \text{ horas} = 1440 \text{ minutos} = 86400 \text{ segundos}$$

5.! Crear un programa que solicite al usuario que ingrese el precio de un producto al inicio del año, y el precio del mismo producto al finalizar el año. El programa debe calcular cuál fue el porcentaje de aumento que tuvo ese producto en el año y mostrarlo por pantalla.

PROGRAMACIÓN DESDE CERO

ESTRUCTURAS DE CONTROL CON PSEINT – ESTRUCTURAS SELECTIVAS



Universidad de
LA PUNTA



GOBIERNO DE
SAN LUIS



EGG



Objetivos de la Guía

En esta guía aprenderemos a:

- Diferenciar estructuras secuenciales de selectivas.
- Armar estructuras condicionales.
- Usar estructuras condicionales.

GUÍA DE ESTRUCTURAS DE CONTROL

En nuestra primera guía de PSeInt nuestros algoritmos consistieron en simples secuencias de instrucciones unas después de otra. Estos algoritmos, con una instrucción detrás de otra se conocen como estructuras secuenciales, pero en nuestros programas existen tareas más complejas que no pueden ser resueltas así, quizás necesitamos repetir una misma instrucción, realizar acciones diferentes en función del valor de una expresión, etc. Para esto existen las estructuras de control.

¿QUÉ ES UNA ESTRUCTURA DE CONTROL?

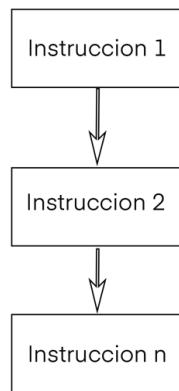
Las **Estructuras de Control** determinan el orden en que deben ejecutarse las instrucciones de un algoritmo, es decir, **si serán recorridas una después de la otra** (estructuras secuenciales), **si habrá que tomar decisiones sobre si ejecutar o no alguna acción** (estructuras selectivas o de decisión) o **si habrá que realizar repeticiones** (estructuras repetitivas). Esto significa que una estructura de control permite que se realicen unas instrucciones y omitir otras, de acuerdo a la evaluación de una condición.

Esto hace que las estructuras de control se puedan dividir en tres:

- Estructuras secuenciales.
- Estructuras selectivas o de decisión.
- Estructuras repetitivas.

¿QUÉ ES UNA ESTRUCTURA SECUENCIAL?

Es la estructura en donde una acción (instrucción) **sigue a otra de manera secuencial**. Las tareas se dan de tal forma **que la salida de una es la entrada de la que sigue** y así en lo sucesivo hasta cumplir con todo el proceso. Esta estructura de control es la más simple, permite que las instrucciones que la constituyen se ejecuten una tras otra en el orden en que se listan.

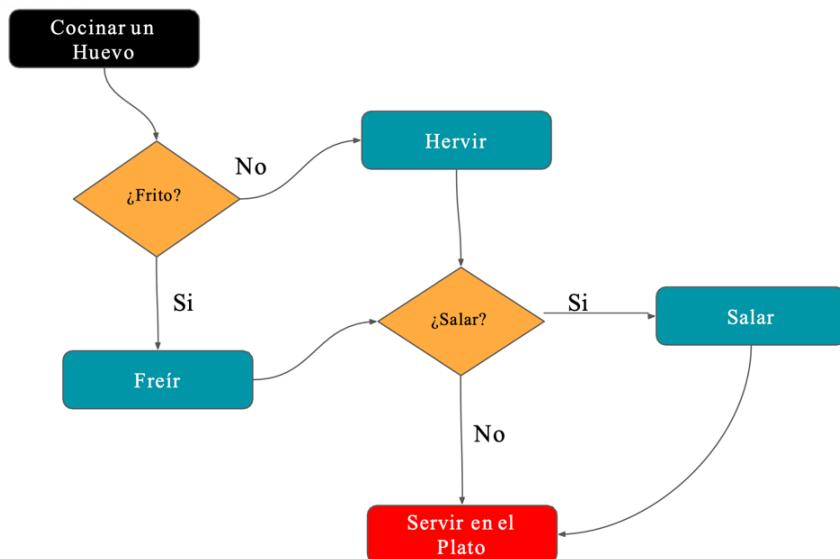


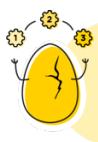
¿QUÉ ES UNA ESTRUCTURA SELECTIVA?

Estas estructuras de control son de gran utilidad para cuando el algoritmo a desarrollar requiera una descripción más complicada que una lista sencilla de instrucciones. Este es el caso cuando existe un número de posibles alternativas que resultan de la evaluación de una determinada condición. Este tipo de estructuras son utilizadas para tomar decisiones lógicas, es por esto que también se denominan **estructuras de decisión o selectivas**.

En estas estructuras, se realiza una **evaluación de una condición y de acuerdo al resultado, el algoritmo realiza una determinada acción**. Las condiciones son especificadas utilizando expresiones lógicas.

Una estructura selectiva, con varias condiciones, sería el ejemplo que usamos en la primera guía de **Cocinar un Huevo**. Las preguntas serían las condiciones a evaluar y de acuerdo a ese resultado realiza una o u otra acción.





El tiempo compartido vale

No tengas miedo en preguntar a quienes van más adelantados. Ellos al explicar reafirman ideas o se dan cuenta que no dominan el concepto. En un futuro, serás tú quien destinará minutos de su tiempo a explicar a un compañero.

¿QUÉ ES UNA CONDICIÓN?

En programación, una condición es toda **sentencia** de la cual se puede determinar **su verdad** (true) o **falsedad** (false). En su gran mayoría, son comparaciones. Por ejemplo, **4 > 5**, ésta sentencia **es una condición porque tiene resultado verdadero o falso**, en este caso falso porque 4 no es mayor a 5. En cambio, la siguiente sentencia, **Escribir "EggEducacion", no es condición** puesto que no hay para comparar, no se puede determinar verdad o falsedad.

Podemos usar como condición el valor de una variable lógica que tengamos definida, ya que es lo mismo: verdadero o falso.

Por lo que una condición sirve para discernir entre una opción u otra, y en el proceso mental normalmente se manifiesta con un "Si"; por ejemplo: Si (va a llover), coge el paraguas.

Para determinar condiciones, precisamos utilizar Operadores.

¿Qué son los operadores?

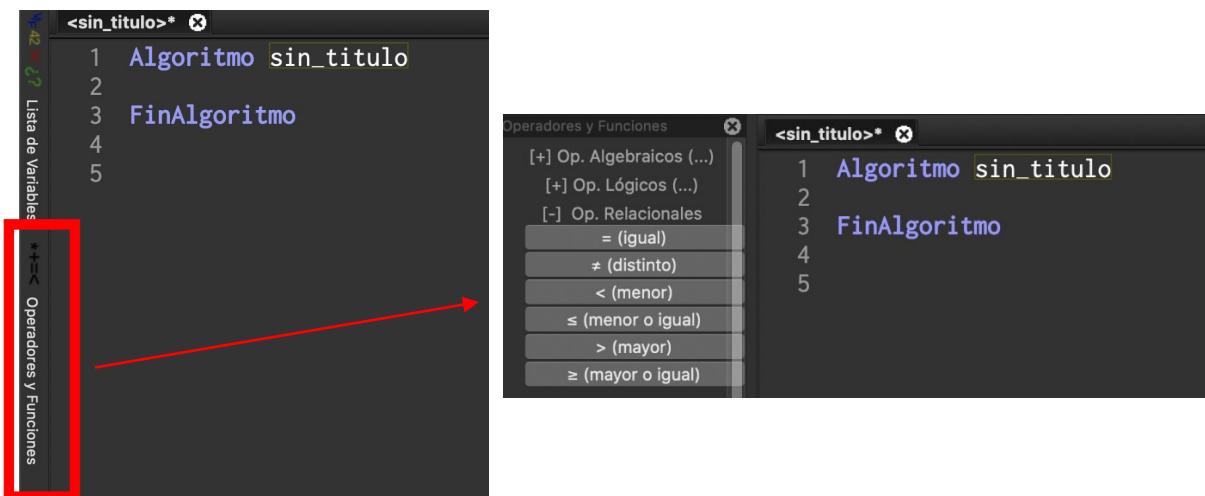
Las condiciones que usaremos en las estructuras selectivas y el resto de nuestras estructuras de control se realizan con la ayuda de los operadores relacionales y lógicos.

OPERADORES RELACIONALES

Los operadores relacionales son símbolos que se usan para comparar dos valores. Si el resultado de la comparación es correcto la expresión considerada es verdadera, en caso contrario es falsa.

Operador	Significado	Ejemplo
Relacionales		
>	Mayor que	3 > 2 // verdadero
<	Menor que	1 < 5 // verdadero
==	Igual que	4 == 4 // verdadero
>=	Mayor o igual que	4 >= 5 // falso
<=	Menor o igual que	'a' <= 'b' // verdadero
<>	Distinto que	10 <> 8 // verdadero

Estos operadores los podemos encontrar en el menú izquierdo de PseInt o podemos cargarlos por teclado.



```
<sin_titulo>*
1 Algoritmo sin_titulo
2
3 FinAlgoritmo
4
5
```

Operadores y Funciones

- [+] Op. Algebraicos (...)
- [+] Op. Lógicos (...)
- [-] Op. Relacionales
 - = (igual)
 - * (distinto)
 - < (menor)
 - \leq (menor o igual)
 - > (mayor)
 - \geq (mayor o igual)

```
<sin_titulo>*
1 Algoritmo sin_titulo
2
3 FinAlgoritmo
4
5
```

OPERADORES LÓGICOS

Estos se utilizan cuando necesitamos las expresiones lógicas con múltiples variantes y nos proporcionan un resultado a partir de que se cumpla o no una cierta condición, estos producen un resultado lógico, y sus operadores son también valores lógicos o asimilables a ellos.

Operador	Significado	Ejemplo
Lógicos		
Y	Conjunción	$(2 < 4 \text{ Y } 3 > 5) // \text{falso}$
O	Disyunción	$(7 \leq 8 \text{ O } 10 \geq 9) // \text{verdadero}$
NO / no	Negación	$\text{no}(1 == 1) // \text{falso}$

Operador Y

Devuelve un valor lógico verdadero si ambas expresiones son verdaderas. En caso contrario el resultado es falso.

Operador O

Este operador devuelve verdadero si alguna de las expresiones es verdadera. En caso contrario devuelve "falso".

Operador NO

Este operador cambia la devolución de una expresión, al caso contrario. Si es verdadero lo hace falso y si es falso lo hace verdadero.

A la hora de trabajar con operadores lógicos, para saber si una expresión lógica nos devuelve como resultado Verdadero o Falso, debemos observar la siguiente **tabla de la verdad**:



Conjunción

A	Operador	B	Resultado
V	Y	V	V
V	Y	F	F
F	Y	V	F
F	Y	F	F

Disyunción

A	Operador	B	Resultado
V	O	V	V
V	O	F	V
F	O	V	V
F	O	F	F

Negación

A	Resultado	B	Resultado
no(V)	F	no(F)	V

Estos operadores también los podemos encontrar en el menú izquierdo de PSeInt o podemos cargarlos por teclado.

The screenshot shows the PSeInt environment. On the left, the 'Operadores y Funciones' menu is open, displaying categories like 'Op. Algebraicos (...)' and 'Op. Lógicos'. Under 'Op. Lógicos', three operators are listed: 'Y (conjunción)', 'O (disyunción)', and 'NO (negación)'. The 'Y (conjunción)' option is highlighted. To the right, a code editor window titled '<sin_titulo>*' contains the following pseudocode:

```

1 Algoritmo sin_titulo
2
3 FinAlgoritmo
4
5

```

Reglas de prioridad:

Las expresiones que tienen dos o más operadores requieren unas reglas matemáticas que permitan determinar el orden de las operaciones, se denominan reglas de prioridad y son:

1. Las operaciones que están encerradas entre paréntesis se evalúan primero. Si existen diferentes paréntesis anidados (interiores unos a otros), las expresiones más internas se evalúan primero.
2. Las operaciones lógicas dentro de una expresión suelen seguir el siguiente orden de prioridad:
 - operador ()
 - operador negación NO
 - operador conjunción Y
 - operador disyunción O

En caso de coincidir varios operadores de igual prioridad en una expresión o sub expresión encerrada entre paréntesis, el orden de prioridad en este caso es de **izquierda a derecha**, y a esta propiedad se denomina asociatividad.



¿NECESITAS UN EJEMPLO?

Vamos a mostrar ejemplos de condiciones tanto con operadores relacionales, como con lógicos

$x==y$, significa "si x es igual a y "

$x>y$, significa "si x es mayor que y "

$x<y$, significa "si x es menor que y "

$x!=y$, significa "si x es distinto de y "

$(x==j) \text{ Y } (x==z)$, significa "si x es igual a j Y x igual a z "

$(x==y) \text{ O } (x==z)$, significa "si x es igual a j O x igual a z "



MANOS A LA OBRA!

EJERCICIO OPERADORES

Vamos a poner en práctica los usos de los operadores, para ello definiremos una variable de tipo lógico y utilizaremos los operadores lógicos y relacionales para otorgarle un valor y mostrarlo por pantalla.

1 Algoritmo PRACTICA_OPERADORES 2 3 Definir bandera Como Logico 4 5 bandera = 4 > 5 6 7 Escribir bandera 8 9 FinAlgoritmo 10 11	 PSelInt - Ejecutando proceso PRACTICA_OPERADORES *** Ejecución Iniciada. *** FALSO *** Ejecución Finalizada. ***
--	--

Prueba tantos operadores lógicos como relacionales quieras. Intenta aplicar las reglas de prioridad y validar varias condiciones a la vez con operadores lógicos.

DETECCIÓN DE ERRORES

Copia y pega este código en tu programa. Deberás corregir los errores hasta lograr el siguiente resultado esperado:

```
Algoritmo PRÁCTICA_OPERADORES
bandera ← 4 < 5
Escribir bandera
```

```
bandera ← (2 12 MOD 2) 0 (NO 3 5 0 32 3^5)
```

Escribir bandera

```
bandera ← 3 > 2 5 >= 3
```

FinAlgoritmo

¿Cuál es el resultado a lograr?

```
*** Ejecución Iniciada. ***
FALSO
VERDADERO
VERDADERO
*** Ejecución Finalizada. ***
```



Revisemos lo aprendido hasta aquí

- Diferenciar estructuras secuenciales de las selectivas.
- Elaboración de condiciones
- Implementación de operadores relacionales y lógicos para la creación de condiciones.

¿CUÁLES SON LAS ESTRUCTURAS SELECTIVAS?

Entendemos que las estructuras selectivas son utilizadas para tomar decisiones lógicas, es por esto que también se denominan **estructuras de decisión o selectivas**. Pero, ¿cuáles son las estructuras selectivas?

Las estructuras selectivas/alternativas pueden ser:

- Simples: Si
- Doble: Si- SiNo
- Múltiples: Según – Si Anidado

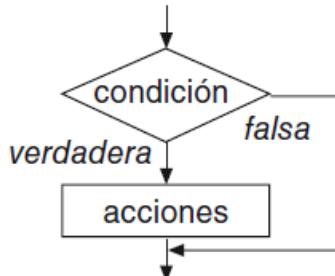


Para evitar errores de sintaxis, recomendamos seleccionar las estructuras selectivas desde el panel de comandos de la derecha en PSeInt.



CONDICIÓN SIMPLE

La estructura alternativa simple si-entonces lleva a cabo una acción siempre y cuando se cumpla una determinada condición.

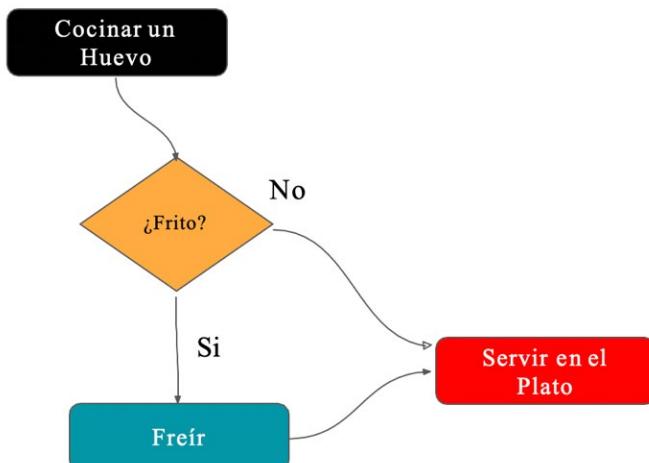


La selección si-entonces evalúa la condición y luego:

- Si la condición es verdadera, ejecuta el bloque de acciones
- Si la condición es falsa, no ejecuta otra opción.



¿NECESITAS UN EJEMPLO?



Si fuera cocinar un huevo, tenemos solo la opción de freírlo y si no lo queremos frito, se va servir crudo en el plato. Esto es una **Condición Simple**.

Condición Simple en PSeInt:

```
Si condicion Entonces
    acciones_por_verdadero
Fin Si
```



Pueden encontrar un ejemplo para descargar de Condición Simple en el Aula Virtual.



MANOS A LA OBRA!

EJERCICIO FACILITADOR

Escriba un programa en donde se le pregunte al estudiante si el día de hoy le toca ser el facilitador de tu equipo. En caso de que sea, que muestre por pantalla el siguiente mensaje: "¡Felicitaciones! Eres el facilitador de tu equipo."

DETECCIÓN DE ERRORES

Copia y pega este código en tu programa. Deberás corregir los errores hasta lograr el siguiente resultado esperado:

```
Algoritmo PRACTICA_CONDICIONAL_SI
```

```
bandera ← 2 <= 1
```

```
Si band Entonces
```

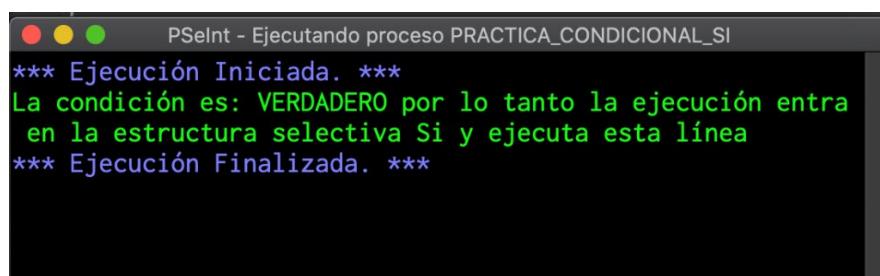
Escribir "La condición es: " " por lo tanto la ejecución entra en la estructura selectiva Si y ejecuta esta línea "

```
Fin Si
```

```
FinAlgoritmo
```

¿Cuál es el resultado a lograr?

No podemos escribir Verdadero en el escribir, deberemos llegar a ese resultado a través del código.



```
PSelnt - Ejecutando proceso PRACTICA_CONDICIONAL_SI
*** Ejecución Iniciada. ***
La condición es: VERDADERO por lo tanto la ejecución entra
en la estructura selectiva Si y ejecuta esta línea
*** Ejecución Finalizada. ***
```



Revisemos lo aprendido hasta aquí

- Crear la Estructura Sí, permitiendo al programa ejecutar un bloque de código si la condición que estableciste es verdadera.

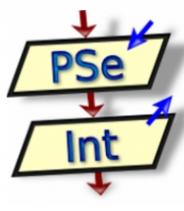
Si no pudiste interiorizar el concepto en su totalidad, **no te preocupes**, más adelante seguiremos trabajando sobre este tema.

PROGRAMACIÓN DESDE CERO

MATERIAL DE TRABAJO

Estructuras de control con Pselnt – Estructuras selectivas

ENCUENTRO 5



Universidad de
LA PUNTA



GOBIERNO DE
SAN LUIS



EGG



Objetivos de la Guía

En esta guía aprenderemos a:

- Diferenciar estructuras secuenciales de selectivas.
- Armar estructuras condicionales.
- Usar estructuras condicionales.

GUÍA DE ESTRUCTURAS DE CONTROL

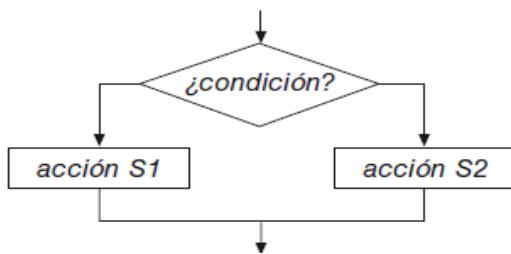
En la guía anterior vimos que las estructuras selectivas son utilizadas para tomar decisiones lógicas, es por esto que también se denominan **estructuras de decisión o selectivas**. Y que existen tres tipos:

- Simples: Si
- Doble: Si- SiNo
- Múltiples: Según – Si Anidado

La última vez **solo vimos las simples**, por lo que hoy veremos una de las dos restantes, las dobles.

CONDICIÓN DOBLE

La estructura anterior es muy limitada y normalmente se necesitará una estructura que permita elegir entre dos opciones o alternativas posibles, en función del cumplimiento o no de una determinada condición. Si la condición es verdadera, se ejecuta la acción S1 y, si es falsa, se ejecuta la acción S2.

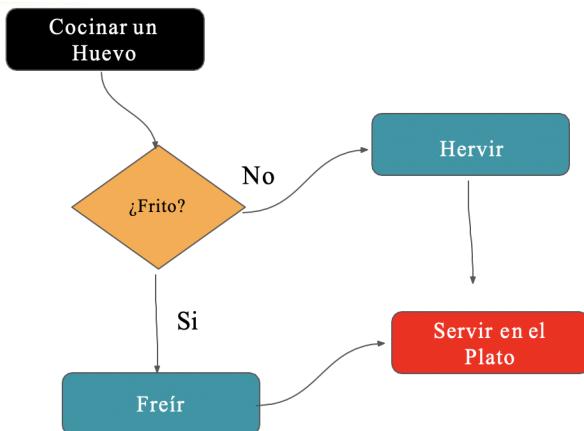


La selección si-entonces-sino evalúa la condición y luego:

- Si la condición es verdadera, ejecuta el bloque de acciones
- Si la condición es falsa, ejecuta el bloque de acciones 2.



¿NECESITAS UN EJEMPLO?

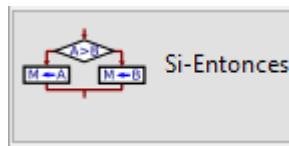


En este caso, si fuera cocinar un huevo, tenemos opción de freírlo y si no lo queremos frito, tendremos la opción de hervirlo. Esto es una **Condición Doble**.

Condición Doble en PSelint

```

Si condicion Entonces
  acciones_por_verdadero
SiNo
  acciones_por_falso
Fin Si
  
```



Pueden encontrar un ejemplo para descargar de Condición Doble en el Aula Virtual.



MANOS A LA OBRA!

EJERCICIO ZOOM

Realizar un programa que pida al usuario el horario en el que se conectó al zoom. Si ese horario está entre la hora de ingreso y la hora de ingreso + los 15' de tardanza, mostrará un mensaje por pantalla que diga “Llegaste a tiempo, tienes presente”. Si el horario de ingreso supera ese límite, se mostrará un mensaje por pantalla que diga “Hoy tendrás tardanza. Recuerda avisarle a tus coaches cuando llegarás tarde. Mañana te esperamos a tiempo, tu participación en el equipo es VITAL”

DETECCIÓN DE ERRORES

Copia y pega este código en tu programa. Deberás corregir los errores hasta lograr el siguiente resultado esperado:

Algoritmo PRÁCTICA CONDICIONAL SINO

Definir bandera Como Logico

bandera = 2 > 3 3 == 3

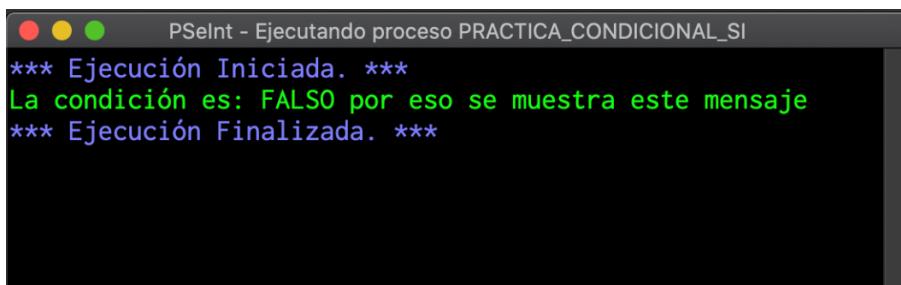
Si bandera Entonces

Escribir "La condición es: " bandera " por eso se muestra este mensaje"

SiNo

Escribir "La condición es: " bandera " por eso se muestra este mensaje"

¿Cuál es el resultado a lograr?



```
PSelnt - Ejecutando proceso PRACTICA_CONDICIONAL_SI
*** Ejecución Iniciada. ***
La condición es: FALSO por eso se muestra este mensaje
*** Ejecución Finalizada. ***
```



Revisemos lo aprendido hasta aquí

- Implementar la Estructura Si-Sino, permitiendo al programa ejecutar un bloque de código si la condición que estableciste es verdadera y otro bloque distinto si la condición es falsa

Si no pudiste interiorizar el concepto en su totalidad, **no te preocupes**, más adelante seguiremos trabajando sobre este tema.

FUNCIONES PSEINT

Además de empezar a implementar las estructuras de control, vamos a empezar a utilizar las **funciones de PSeInt**. Las funciones, son herramientas que nos proporciona PSeInt y cumplen el propósito de *ayudarnos a resolver ciertos problemas*. Supongamos que tenemos que calcular la raíz cuadrada de un número, PSeInt cuenta con una función que, pasándole un número, nos devuelve el resultado de su raíz cuadrada. Ese resultado que devuelve, se lo podemos asignar a una variable o lo podemos concatenar con un escribir para mostrar el resultado sin la necesidad de una variable.

También, las *funciones se pueden utilizar dentro de cualquier expresión, de cualquier estructura*, y cuando se evalúe la misma, se reemplazará por el resultado correspondiente.

Tenemos dos tipos de funciones, funciones matemáticas y funciones de cadenas de texto. Las funciones matemáticas, reciben un sólo parámetro de tipo numérico y devolverán un solo valor de tipo numérico. Las funciones de cadenas, en cambio, reciben un solo parámetro de tipo cadena, pero pueden devolver un valor de tipo cadena o de tipo numérico según la función que se use.

Funciones	Significado
RC(número)	Devuelve la raíz cuadrada del número.
ABS(número)	Devuelve el valor absoluto del número
LN(número)	Devuelve el logaritmo natural del número
EXP(número)	Devuelve la función exponencial del número.
SEN(número)	Devuelve el seno de número.
COS(número)	Devuelve el coseno de número.
TAN(número)	Devuelve la tangente de número.
ASEN(número)	Devuelve el arcoseno de número.
ACOS(número)	Arcocoseno de x
ATAN(número)	Arcotangente de x
MOD	Devuelve el módulo (resto de la división entera).
TRUNC(número)	Trunca el valor x (parte entera de x)
REDOND(número)	Redondea al valor más cercano a x
AZAR(número)	Entero aleatorio entre 0 y x -1
ALEATORIO(min,max)	Entero aleatorio entre valor mínimo y máximo



¿NECESITAS UN EJEMPLO?

Escribir "Raíz cuadrada de 9: " `rc(9)`

Escribir "Resto de 4/2: " `4 MOD 2`

Escribir "Valor absoluto de -3: " `abs(-3)`

Escribir "Seno de 90 grados: " `sen(90 * PI / 180)`

Escribir "Truncamos 3.7: " `trunc(3.7)`

Escribir "Redondeamos 2.7: " `redon(2.7)`

Escribir "Un número al azar del 0 al 9: " `azar(10)`

Escribir "Un número al azar entre 10 y 20: " `aleatorio(10,20)`

Del código anterior los resultados serían:

Raíz cuadrada de 9: **3**

Resto e 4/2: **0**

Valor absoluto de -3: **3**

Seno de 90 grados: **1**

Truncamos 3.7: **3**

Redondeamos 2.7: **3**

Un número al azar del 0 al 9: **6**

Un número al azar entre 10 y 20: **14**



Pueden encontrar un ejemplo para descargar de Funciones Matemáticas en el Aula Virtual.

FUNCIONES CADENAS DE TEXTO

Algunas funciones de cadenas de texto utilizan las posiciones de cada letra de una cadena. Esto significa que, si tengo la palabra Hola, la cadena tendrá 4 posiciones, en PSeInt las posiciones de las letras arrancan en 0. Entonces para la cadena Hola, nuestras posiciones serían: 0: **H**, 1: **o**, 2: **I** y 3: **a**.

Funciones	Significado
Longitud(cadena)	Devuelve la cantidad de letras que compone la cadena.
Mayusculas(cadena)	Devuelve una copia de la cadena con todas sus letras en mayúsculas.

Minusculas(cadena)	Devuelve una copia de la cadena con todas sus letras en minúsculas.
Subcadena(cadena, posición_inicial, posición_final)	Devuelve una nueva cadena que consiste en la parte de la cadena que va desde la posición pos_inicial hasta la posición pos_final.
Concatenar(cadena, cadena2)	Devuelve una nueva cadena que resulta de unir las cadenas cadena1 y cadena2.
ConvertirANumero(cadena)	Recibe una cadena compuesta de números y devuelve la cadena como una variable numérica.
ConvertirACadena(cadena)	Recibe un número y devuelve una variable cadena de caracteres de dicho número.



¿NECESITAS UN EJEMPLO?

Definir cadena1,cadena2 como cadena

cadena1 = "programacion"

cadena2 = "EGG"

Escribir "La longitud de cadena1 es: " **longitud**(cadena1)

Escribir "El primer carácter de cadena1 es: " **subcadena**(cadena1,0,0)

Escribir "La cadena1 en mayúsculas es: " **mayusculas**(cadena1)

Escribir "La cadena2 en minusculas es: " **minusculas**(cadena2)

Escribir "La cadena concatenada queda como: " **concatenar**(cadena1," es muy interesante")

Escribir "La cadena convertida a numero queda:" **convertirANumero**("10")

Del código anterior los resultados serían:

La longitud de cadena1 es: **12**

El primer carácter de cadena1 es: **p**

La cadena1 en mayúsculas es: **PROGRAMACION**

La cadena2 en minúsculas es: **egg**

La cadena concatenada queda como: **programacion es muy interesante**

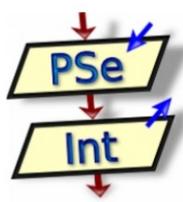
La cadena convertida a numero queda: **10**



Pueden encontrar un ejemplo para descargar de Funciones de Cadenas de Texto en el Aula Virtual.

PROGRAMACIÓN DESDE CERO

ESTRUCTURAS DE CONTROL CON PSEINT – ESTRUCTURAS SELECTIVAS



Universidad de
LA PUNTA



GOBIERNO DE
SAN LUIS



EGG



Objetivos de la Guía

En esta guía aprenderemos a:

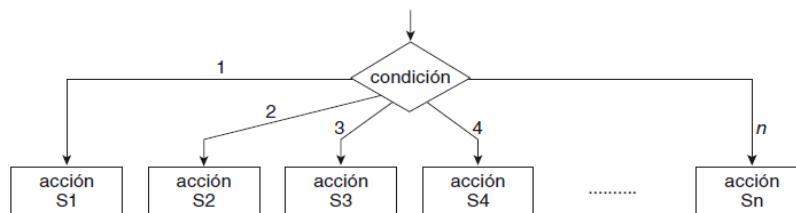
- Diferenciar estructuras secuenciales de selectivas.
- Armar estructuras condicionales.
- Usar estructuras condicionales.

GUÍA DE ESTRUCTURAS DE CONTROL

CONDICIÓN MÚLTIPLE

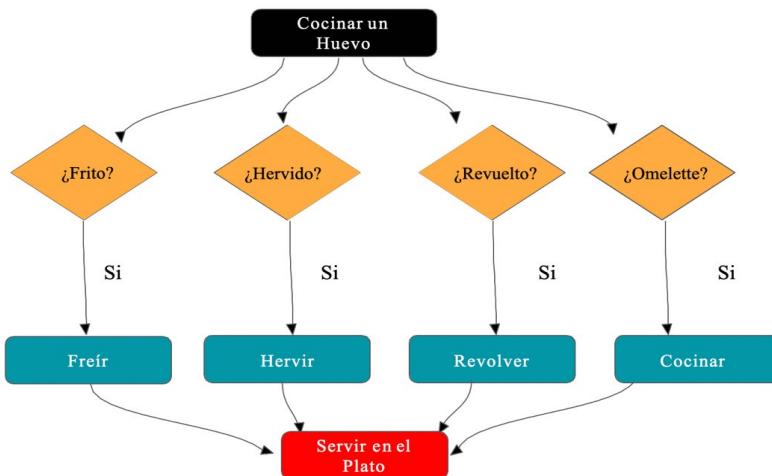
Muchas veces vamos a tener más de dos alternativas para elegir, o una variable que puede tomar varios valores. Para solucionar esto, usamos la condición múltiple. En esta estructura, se evalúa una condición o expresión que puede tomar n valores. Según el valor que la expresión tenga en cada momento se ejecutan las acciones correspondientes al valor.

La estructura de decisión múltiple evaluará una expresión que podrá tomar n valores distintos, 1, 2, 3, 4, ..., n. Según el valor que elija en la condición, se realizará una de las n acciones, o lo que es igual, el flujo del algoritmo seguirá un determinado camino entre los n posibles. Por ejemplo, si tenemos un sistema de notas, donde 6 es desaprobado, 7 es aprobado, 9 es sobresaliente y 10 es excelente. Al tener un valor que puede dar distintas alternativas, usamos la condición múltiple.





¿NECESITAS UN EJEMPLO?



Condición Multiple en PSeInt:



```

Segun variable_de_cualquier_tipo_de_dato Hacer
  opcion_1:
    secuencia_de_acciones_1
  opcion_2:
    secuencia_de_acciones_2
  opcion_3:
    secuencia_de_acciones_3
  De Otro Modo:
    secuencia_de_acciones_dom
Fin Segun
  
```

Este problema, se podría resolver por estructuras alternativas simples o dobles, anidadas o en cascada; sin embargo, este método si el número de alternativas es grande puede plantear serios problemas de escritura del algoritmo y naturalmente de legibilidad.



Cuando el valor de la variable que se **evalúa no coincide con ninguno de los valores que se evalúa**, entonces se ejecutan las acciones dentro del bloque “De Otro Modo” (secuencia_de_acciones_dom), el cual equivale a realizar un “Sino” dentro de las estructuras condicionales.

Además, pueden encontrar un ejemplo para descargar de Condición Multiple en Aula Virtual.



¿NECESITAS UN **EJEMPLO?**

```

1  Algoritmo PRACTICA_CONDICIONAL_SEGUN
2    Definir eleccion Como Caracter
3    Escribir "Ingrese una opción para cocinar su huevo"
4    Escribir "A- FRITO"
5    Escribir "B- HERVIDO"
6    Escribir "C- REVUELTO"
7    Escribir "D- OMELETTE"
8    Leer eleccion
9    Segun eleccion Hacer
10      "A":
11        Escribir "Su huevo se servirá FRITO"
12      "B":
13        Escribir "Su huevo se servirá HERVIDO"
14      "C":
15        Escribir "Su huevo se servirá REVUELTO"
16      "D":
17        Escribir "Su huevo se servirá OMELETTE"
18    De Otro Modo:
19      Escribir "La opción ingresada no está entre las ofrecidas"
20  Fin Segun
21 FinAlgoritmo
22

```

PSeInt - Ejecutando proceso PRACTICA_CONDICIONAL_SEGUN
***** Ejecución Iniciada. *****
 Ingrese una opción para cocinar su huevo
 A- FRITO
 B- HERVIDO
 C- REVUELTO
 D- OMELETTE
 > A
 Su huevo se servirá FRITO
***** Ejecución Finalizada. *****

No cerrar esta ventana Siempre visible

Como podemos ver en el ejemplo, le damos al usuario varias opciones para elegir. El carácter que ingrese será analizado en la línea 9, y lo va comparando con las opciones disponibles. Si encuentra alguna coincidencia ejecutará las líneas de código dentro de esa opción, caso contrario se ejecutará el "De otro modo"



MANOS A LA OBRA!

EJERCICIO MESES

Ingresar un número del 1 – 12 y mostrar el mes del año que corresponde, si el número ingresado no es correcto mostrar un "mensaje de error".

DETECCIÓN DE ERRORES

Copia y pega este código en tu programa. Deberás corregir los errores hasta lograr el siguiente resultado esperado:

```

Algoritmo PRACTICA_SEGUN
Definir num Como Caracter
Escribir "Ingrese un número entre 1 y 3"
Según Hacer
  1
    Escribir "Elegiste la opción 1"
  2:
    Escribir "Elegiste la opción 1"
  3:

```

Escribir Elegiste la opción 1

De Otro Mod

Escribir "No elegiste la opción 1, ni 2, ni 3"

FinAlgoritmo

¿Cuál es el resultado a lograr?

```
*** Ejecución Iniciada. ***
Ingrese un número entre 1 y 3
> 3
Elegiste la opción 3
*** Ejecución Finalizada. ***
```



Revisemos lo aprendido hasta aquí

Implementación Estructura Según, permitiendo al programa ejecutar un bloque de código según varias condiciones, en vez de tener una sola condición y un bloque de código para el verdadero y el falso, acá podemos tener varias condiciones distintas y un bloque de código para cada opción. Y, además, tenemos el de otro modo para el caso que sean falsas todas las opciones.

Si no pudiste interiorizar el concepto en su totalidad, **no te preocupes**, más adelante seguiremos trabajando sobre este tema.

CONDICIONALES ANIDADOS O EN CASCADA

Es posible también utilizar la instrucción *Si* para diseñar estructuras de selección que contengan más de dos alternativas. Por ejemplo, una estructura *Si-entonces* puede contener otra estructura *Si-entonces*, y esta estructura *Si-entonces* puede contener otra, y así sucesivamente cualquier número de veces; a su vez, dentro de cada estructura pueden existir diferentes acciones, a esto se le llama condicionales anidados o en cascada.

CONDICIONALES ANIDADOS EN PSEINT:

```

Si expresion_logica1 Entonces
    acciones_por_verdadero1
Sino
Si expresion_logica2 Entonces
    acciones_por_verdadero2
Sino
    Si expresion_logica4 Entonces
        acciones_por_verdadero3
    Sino
        acciones_por_falso
Fin Si
Fin Si
Fin Si

```



¿NECESITAS UN EJEMPLO?

```

1 Algoritmo SiAnidado
2     Definir nota Como Entero
3     Escribir "Ingrese su nota"
4     Leer nota
5     /// Anidamos los si para tener una accion para las distintas posibilidades
6     Si nota ≤ 6 Entonces
7         Escribir "Desaprobo"
8     SiNo
9         Si nota = 7
10            Escribir "Aprobo"
11        SiNo
12            Si nota = 8
13                Escribir "Muy bien"
14            SiNo
15                Si nota = 9 Entonces
16                    Escribir "Sobresaliente"
17                SiNo
18                    Si nota = 10
19                        Escribir "Excelente"
20
21                    FinSi
22
23            FinSi
24
25        Fin Si
26    FinSi
27
28 FinAlgoritmo

```



Pueden encontrar un ejemplo para descargar de Condicionales Anidados en el Aula Virtual.



MANOS A LA OBRA!

EJERCICIO DESAYUNO

Es tu turno, diseña un condicional anidado que le pregunte al usuario si quiere tomar té o café y en caso de que quiera tomar café, preguntar si solo o cortado y en caso de ser cortado, si prefiere leche vegetal.

DETECCIÓN DE ERRORES

Copia y pega este código, que muestra el mayor de 3 numeros en tu programa. Deberás corregir los errores hasta lograr el siguiente resultado esperado:

Algoritmo Correccion_SiAnidado

Definir n1, n2 Como Entero

Escribir "MOSTRAR EL MAYOR DE 3 NÚMEROS"

Escribir "INGRESE NÚMERO 01 : "

n1

Escribir "INGRESE NÚMERO 02 : "

Escribir "INGRESE NÚMERO 03 : "

Leer n3

Si (n1 > n2 n1 n3) Entonces

 Escribir "MAYOR ES ", n1

No

 Si (n2 n1 n2 n3) Entonces

 Escribir "MAYOR ES ", n2

 Escribir "MAYOR ES ", n3

 FinSi

FinSi

FinAlgoritmo

¿Cuál es el resultado a lograr?

```
● ● ● PSeInt - Ejecutando proceso CORRECCION_SIANIDADO
*** Ejecución Iniciada. ***
MOSTRAR EL MAYOR DE 3 NÚMEROS
INGRESE NÚMERO 01 :
> 3
INGRESE NÚMERO 02 :
> 4
INGRESE NÚMERO 03 :
> 9
El mayor es 9
*** Ejecución Finalizada. ***
```



Revisemos lo aprendido hasta aquí

- Identificar, diferenciar y elaborar condicionales anidados. Donde el programa puede evaluar una condición y en base a ella, evaluar otras condiciones tanto si son verdaderas o falsas y ejecutar bloques de códigos cuando esas condiciones sean verdaderas.

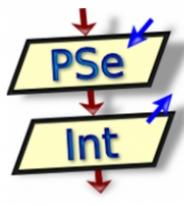
Si no pudiste interiorizar el concepto en su totalidad, **no te preocupes**, más adelante seguiremos trabajando sobre este tema.

PROGRAMACIÓN DESDE CERO

MATERIAL DE TRABAJO

Estructuras de control con Pselnt – Estructuras repetitivas

ENCUENTRO 7





Objetivos de la Guía

En esta guía aprenderemos a:

- Armar estructuras repetitivas.
- Usar estructuras repetitivas.

GUÍA ESTRUCTURAS REPETITIVAS

Vamos a continuar con las estructuras de control, pero ahora vamos a presentar un nuevo tipo de estructura. En los encuentros anteriores, nos centramos en las estructuras selectivas, en estos próximos encuentros veremos **las estructuras repetitivas**.

¿QUÉ SON LAS ESTRUCTURAS REPETITIVAS?

Durante el proceso de creación de programas, es muy común encontrarse con que una operación o conjunto de operaciones deben repetirse muchas veces. Para ello es importante conocer las estructuras de algoritmos que permiten repetir una o varias acciones, un número determinado de veces.

Las estructuras que repiten una secuencia de instrucciones un número determinado de veces se denominan bucles, y se denomina iteración al hecho de repetir la ejecución de una secuencia de acciones.

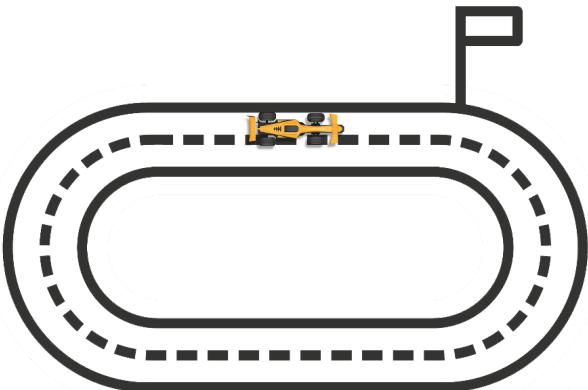
Todo bucle tiene que llevar asociada una condición, que es la que va a determinar cuándo se repite el bucle y cuando deja de repetirse.

Hay distintos tipos de bucles:

- Mientras
- Hacer Mientras
- Para



¿NECESITAS UN EJEMPLO?

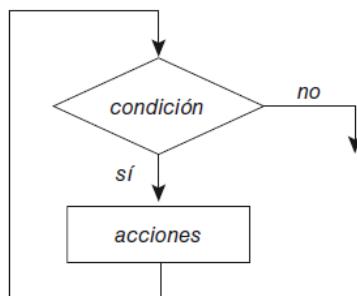


Este ejemplo nos muestra el concepto de bucle. Este auto de F1 debe dar vueltas a la pista hasta que la cantidad de vueltas realizadas sea igual a la cantidad total de vueltas que exige la carrera. Por lo tanto, debe repetir la vuelta tantas veces hasta que termine la carrera.

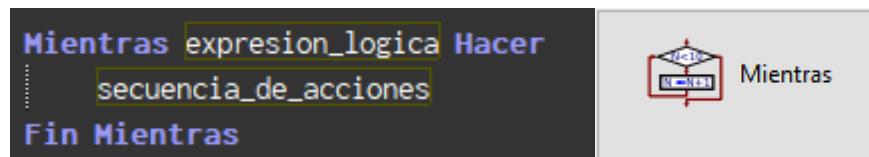


ESTRUCTURA MIENTRAS

Esta estructura repetitiva **Mientras**, es en la que el cuerpo del bucle se repite siempre que se cumpla una determinada *condición*. Cuando se ejecuta la instrucción mientras, la primera cosa que sucede es que se evalúa la condición (una expresión lógica). Si se evalúa *falsa*, no se toma ninguna acción y el programa prosigue con la siguiente instrucción. Si la expresión lógica es *verdadera*, entonces se ejecuta el cuerpo del bucle, después de lo cual se evalúa de nuevo la expresión lógica. Este proceso se repite una y otra vez mientras la expresión lógica (condición) sea verdadera, para salir del bucle la condición debe ser falsa.



Estructura Mientras en PSeInt



Regla práctica

En las expresiones lógicas es conveniente usar comparaciones mayor o menor en lugar de comparaciones de igualdad o desigualdad. En el caso de la codificación en un lenguaje de programación, esta regla debe seguirse rígidamente en el caso de comparación de números reales, ya que como esos valores se almacenan en cantidades aproximadas las comparaciones de igualdad de valores reales normalmente plantean problemas. Siempre que realice comparaciones de números reales use las relaciones <, <=, > o >=.



¿NECESITAS UN EJEMPLO?

```
1 Algoritmo EjemploMientras
2
3     Definir nota Como Entero
4
5     Escribir "Ingrese una nota válida"
6     Leer nota
7
8     Mientras nota < 0 o nota > 10
9
10        Escribir "Ingrese la nota nuevamente"
11        Leer nota
12
13    FinMientras
14
15    Escribir "La nota es correcta"
16
17 FinAlgoritmo
18
```

En este bucle analizamos si la nota ingresada **no** es válida, es decir si **no** está en el rango entre 0 y 10. Si la nota se sale del rango la condición será verdadera y se ejecutará el código dentro del bucle. Si la condición es falsa, no se ejecuta el código dentro del bucle y va directo a la línea 15.



Pueden encontrar un ejemplo para descargar del Bucle Mientras en Aula Virtual.



MANOS A LA OBRA!

EJERCICIO VOCAL SECRETA

Diseña un programa que guarde una vocal secreta en una variable, debemos pedirle al usuario que intente adivinar la vocal secreta, e intentará tantas veces como sea necesario hasta que la adivine.

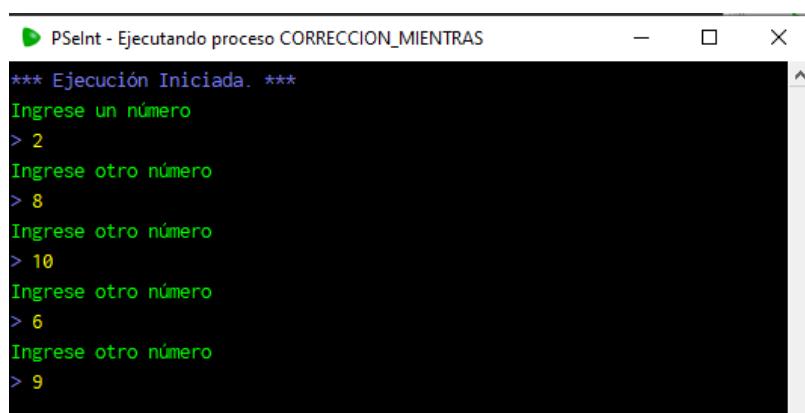
DETECCIÓN DE ERRORES

Copia y pega este código en tu programa. Deberás corregir los errores hasta lograr el siguiente resultado esperado:

```
Algoritmo Correccion_Mientras
Definir num Como Entero
//El programa ingresará números mientras sean PARES
Escribir "Ingrese un número"
Leer num
Mientras num 2 == 0 Hacer
    Escribir "Ingrese otro número"

FinAlgoritmo
```

¿Cuál es el resultado a lograr?



```
PSelnt - Ejecutando proceso CORRECCION_MIENTRAS
*** Ejecución Iniciada. ***
Ingrese un número
> 2
Ingrese otro número
> 8
Ingrese otro número
> 10
Ingrese otro número
> 6
Ingrese otro número
> 9
```

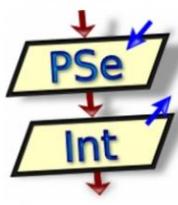


Revisemos lo aprendido hasta aquí

Definir, implementar y diferenciar la estructura **MIENTRAS**. Sabiendo que es la estructura que **PRIMERO** valida la condición y luego ejecuta el código repetidamente **MIENTRAS** la condición sea verdadera.

PROGRAMACIÓN DESDE CERO

ESTRUCTURAS DE CONTROL CON PSEINT – ESTRUCTURAS REPETITIVAS



Universidad de
LA PUNTA



GOBIERNO DE
SAN LUIS

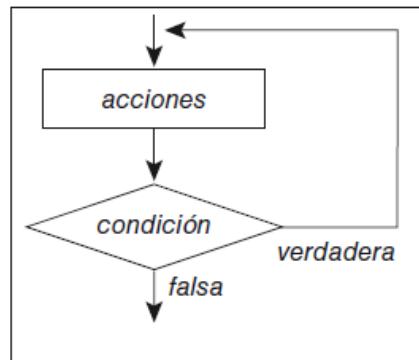


EGG

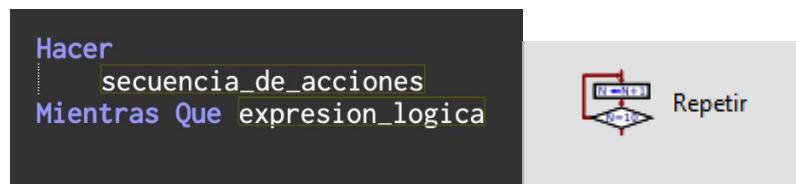


ESTRUCTURA HACER- MIENTRAS

Esta estructura es muy similar a la anterior, sólo que a diferencia del **Mientras** el contenido del bucle **Hacer-Mientras** se ejecuta siempre al menos una vez, ya que la evaluación de la condición lógica se encuentra al final del bucle. De esta forma garantizamos que las acciones dentro de este bucle sean llevadas a cabo al menos una vez, incluso aunque la expresión lógica sea falsa.



Estructura Hacer-Mientras en PseInt:



Regla práctica

El bucle *hacer-mientras* se termina de ejecutar cuando el valor de la condición es falso. La elección entre un bucle mientras y un bucle hacer-mientras depende del problema de cómputo a resolver. En la mayoría de los casos, el bucle mientras es la elección correcta. Por ejemplo, si el bucle se utiliza para recorrer una lista de números (o una lista de cualquier tipo de objetos), la lista puede estar vacía, en cuyo caso las sentencias del bucle nunca se ejecutarán. Si se aplica un bucle hacer-mientras nos conduce a un código de errores.



¿NECESITAS UN EJEMPLO?

```
1  Algoritmo EjemploMientraQue
2
3      Definir nota como entero
4
5
6      /// En este bucle buscamos las notas que esten fuera de 0 o 10,
7      /// para que el bucle de verdadero y se pida la nota de nuevo.
8      /// Nosotros no estamos buscando que ingrese la nota de nuevo
9      /// cuando sea correcta, sino cuando sea incorrecta
10     Hacer
11         /// A diferencia del mientras pedimos la nota adentro,
12         /// ya que se el bucle se corre por lo menos una vez
13         Escribir "Ingrese una nota valida"
14         Leer nota
15
16     Mientras Que nota < 0 o nota > 10
17         /// Ponemos la condicion al final
18
19
20         Escribir "La nota es correcta"
21
22     FinAlgoritmo
23
```

▶ PSeInt - Ejecutando proceso EJEMPLOMIENTRAQUE
*** Ejecución Iniciada. ***
Ingrese una nota valida
> 12
Ingrese una nota valida
> 12
Ingrese una nota valida
> 15
Ingrese una nota valida
> 6
La nota es correcta
*** Ejecución Finalizada. ***



Pueden encontrar un ejemplo para descargar del Bucle Hacer-Mientras en Aula Virtual.



MANOS A LA OBRA!

EJERCICIO VOCAL SECRETA – PARTE 2

Vamos a hacer nuevamente el ejercicio de la vocal misteriosa, pero esta vez con una estructura Hacer-Mientras. ¿Puedes notar cuál es la diferencia entre ambas estructuras?

DETECCIÓN DE ERRORES

Copia y pega este código en tu programa. Deberás corregir los errores hasta lograr el siguiente resultado esperado:

```
Algoritmo Correccion_HacerMientras
```

```
    num Como Entero
```

```
    Repet
```

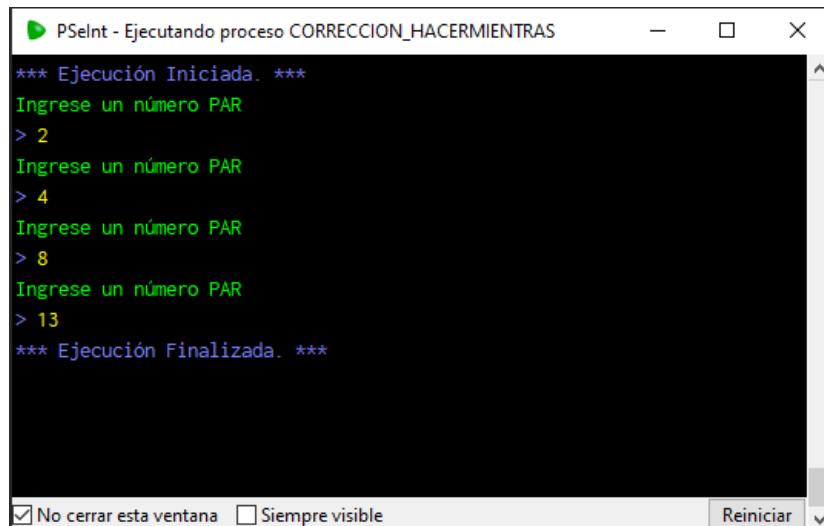
```
        Escribir "Ingrese un número PAR"
```

```
        Leer num
```

```
        Mientras Qe num MOD 2 == 0
```

```
    FinAlgoritmo
```

¿Cuál es el resultado a lograr?



```
PSelint - Ejecutando proceso CORRECCION_HACERMIENTRAS
*** Ejecución Iniciada. ***
Ingrese un número PAR
> 2
Ingrese un número PAR
> 4
Ingrese un número PAR
> 8
Ingrese un número PAR
> 13
*** Ejecución Finalizada. ***
```

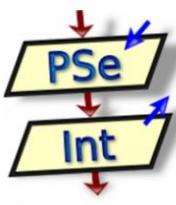


Revisemos lo aprendido hasta aquí

Identificar, diferenciar y elaborar una estructura HACER MIENTRAS, cuya primera validación de la condición se da DESPUÉS de la primera ejecución del bloque de código. Es decir que esta estructura siempre se ejecutará AL MENOS UNA VEZ.

PROGRAMACIÓN DESDE CERO

ESTRUCTURAS DE CONTROL CON PSEINT – ESTRUCTURAS REPETITIVAS



Universidad de
LA PUNTA



GOBIERNO DE
SAN LUIS

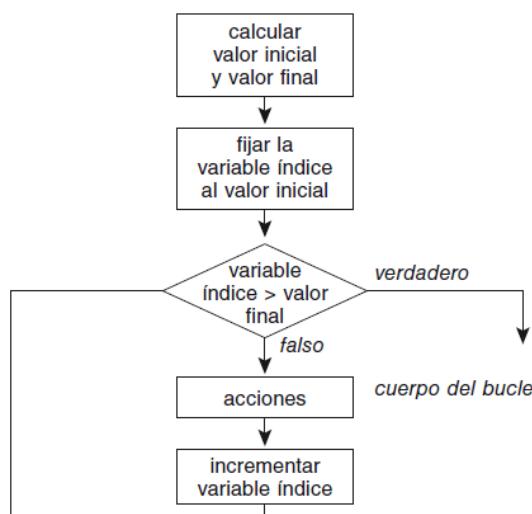


EGG

ESTRUCTURA PARA

La estructura **Para** es un poco más compleja que las anteriores y nos permite ejecutar un conjunto de acciones, para cada paso de un conjunto de elementos. Su implementación depende del lenguaje de programación, pero en términos generales podemos identificar tres componentes: la *inicialización*, *finalización* y el *incremento*.

La estructura **Para** comienza con un valor inicial de una variable llamada índice y las acciones especificadas se ejecutan x cantidad de veces, hasta que el valor índice llegue al valor final, *a menos que el valor inicial sea mayor que el valor final*. La variable índice se incrementa en uno y si este nuevo valor no excede al final, se ejecutan de nuevo las acciones. Por consiguiente, las acciones específicas en el bucle se ejecutan para cada valor de la variable índice desde el valor inicial hasta el valor final con el incremento de uno en uno.

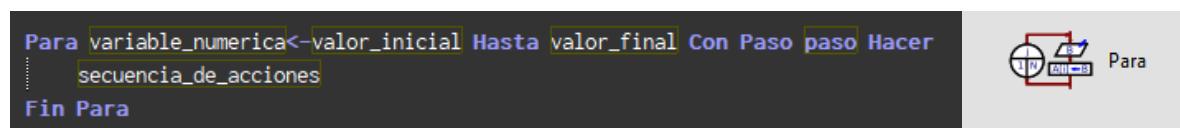


Estructura Para en PSeInt:

```

Para variable_numerica<-valor_inicial Hasta valor_final Con Paso paso Hacer
    secuencia_de_acciones
Fin Para

```



El incremento de la variable índice (variable_numerica) siempre es 1 si no se indica expresamente lo contrario en el valor de *con paso*. Dependiendo del tipo de lenguaje, es posible que el incremento sea distinto de uno, positivo o negativo. La variable índice o de control (variable_numerica) normalmente será de tipo entero y es normal emplear como nombres las letras i, j, k.

Si el valor_inicial de la variable índice es menor que el valor_final, los incrementos, es decir los pasos, deben ser positivos, ya que en caso contrario la secuencia de acciones no se ejecutaría. De igual modo, si el valor_iniciales es mayor que el valor_final, el paso debe ser en este caso negativo, es decir, decremento.



¿NECESITAS UN EJEMPLO?

```
1 Algoritmo EjemploPara
2
3     Definir i Como Entero
4
5     Para i <- 1 Hasta 10 Con Paso 1 Hacer
6         .
7             Escribir "La tabla del 2 es:" i * 2
8         .
9     Fin Para
10
11 FinAlgoritmo
12
13
```

```
PSelnt - Ejecutando proceso EJEMPLOPARA
*** Ejecución Iniciada. ***
La tabla del 2 es:2
La tabla del 2 es:4
La tabla del 2 es:6
La tabla del 2 es:8
La tabla del 2 es:10
La tabla del 2 es:12
La tabla del 2 es:14
La tabla del 2 es:16
La tabla del 2 es:18
La tabla del 2 es:20
*** Ejecución Finalizada. ***
```



Pueden encontrar un ejemplo para descargar del Bucle Para en el Aula Virtual.



MANOS A LA OBRA!

EJERCICIO NUMERO MAYOR

Escribir una estructura PARA que le solicite al usuario varios números y al finalizar muestre el mayor número ingresado.

DETECCIÓN DE ERRORES

Copia y pega este código en tu programa. Deberás corregir los errores hasta lograr el siguiente resultado esperado:

```
Algoritmo correccion_Para
Para <-0 Hasta Con Paso Hacer
    Escribir "Imprimimos el valor de i"
    Escribir i
Fin Para
FinAlgoritmo
```

¿Cuál es el resultado a lograr?

PSelnt - Ejecutando proceso SIN_TITULO

```
Imprimimos el valor de i
2
Imprimimos el valor de i
4
Imprimimos el valor de i
6
Imprimimos el valor de i
8
Imprimimos el valor de i
10
Imprimimos el valor de i
12
*** Ejecución Finalizada. ***
```

No cerrar esta ventana Siempre visible Reiniciar ▼

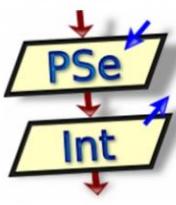


Revisemos lo aprendido hasta aquí

Identificar, construir y utilizar una estructura PARA. En esta estructura tenemos el control absoluto de las repeticiones, ya que podemos determinar la inicialización, límite y aumento del valor de i.

PROGRAMACIÓN DESDE CERO

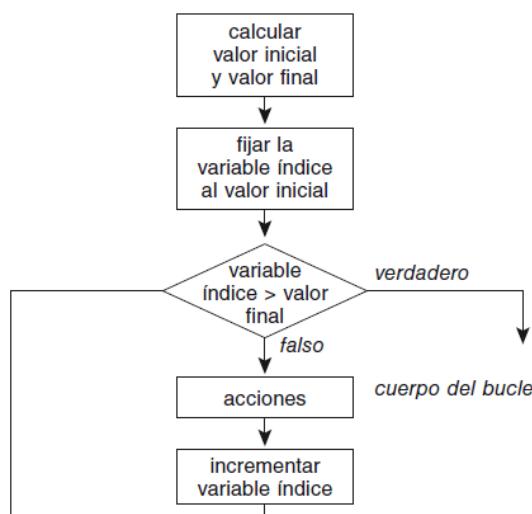
ESTRUCTURAS DE CONTROL CON PSEINT – ESTRUCTURAS REPETITIVAS



ESTRUCTURA PARA

La estructura **Para** es un poco más compleja que las anteriores y nos permite ejecutar un conjunto de acciones, para cada paso de un conjunto de elementos. Su implementación depende del lenguaje de programación, pero en términos generales podemos identificar tres componentes: la *inicialización*, *finalización* y el *incremento*.

La estructura **Para** comienza con un valor inicial de una variable llamada índice y las acciones especificadas se ejecutan x cantidad de veces, hasta que el valor índice llegue al valor final, *a menos que el valor inicial sea mayor que el valor final*. La variable índice se incrementa en uno y si este nuevo valor no excede al final, se ejecutan de nuevo las acciones. Por consiguiente, las acciones específicas en el bucle se ejecutan para cada valor de la variable índice desde el valor inicial hasta el valor final con el incremento de uno en uno.

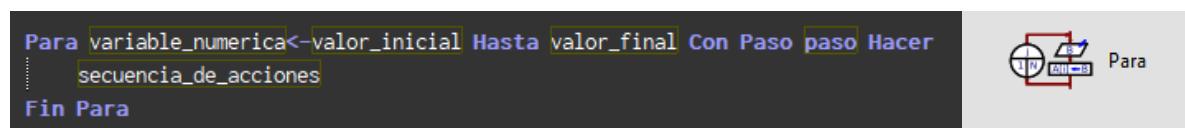


Estructura Para en PSeInt:

```

Para variable_numerica<-valor_inicial Hasta valor_final Con Paso paso Hacer
    secuencia_de_acciones
Fin Para

```



A screenshot of the PSeInt Integrated Development Environment (IDE) showing a block of pseudocode. The code consists of a 'Para' block with 'variable_numerica' set to 'valor_inicial', 'Hasta' 'valor_final', 'Con Paso' 'paso', and 'Hacer' a sequence of actions. Below the 'Hacer' keyword, there is a 'Fin Para' keyword. To the right of the code area, there is a graphical icon of a computer monitor with a circular arrow and the word 'Para' next to it.

El incremento de la variable índice (variable_numerica) siempre es 1 si no se indica expresamente lo contrario en el valor de *con paso*. Dependiendo del tipo de lenguaje, es posible que el incremento sea distinto de uno, positivo o negativo. La variable índice o de control (variable_numerica) normalmente será de tipo entero y es normal emplear como nombres las letras i, j, k.

Si el valor_inicial de la variable índice es menor que el valor_final, los incrementos, es decir los pasos, deben ser positivos, ya que en caso contrario la secuencia de acciones no se ejecutaría. De igual modo, si el valor_iniciales es mayor que el valor_final, el paso debe ser en este caso negativo, es decir, decremento.



¿NECESITAS UN EJEMPLO?

```
1 Algoritmo EjemploPara
2
3     Definir i Como Entero
4
5     Para i <- 1 Hasta 10 Con Paso 1 Hacer
6         .
7             Escribir "La tabla del 2 es:" i * 2
8         .
9     Fin Para
10
11 FinAlgoritmo
12
13
```

```
PSelnt - Ejecutando proceso EJEMPLOPARA
*** Ejecución Iniciada. ***
La tabla del 2 es:2
La tabla del 2 es:4
La tabla del 2 es:6
La tabla del 2 es:8
La tabla del 2 es:10
La tabla del 2 es:12
La tabla del 2 es:14
La tabla del 2 es:16
La tabla del 2 es:18
La tabla del 2 es:20
*** Ejecución Finalizada. ***
```



Pueden encontrar un ejemplo para descargar del Bucle Para en el Aula Virtual.



MANOS A LA OBRA!

EJERCICIO NUMERO MAYOR

Escribir una estructura PARA que le solicite al usuario varios números y al finalizar muestre el mayor número ingresado.

DETECCIÓN DE ERRORES

Copia y pega este código en tu programa. Deberás corregir los errores hasta lograr el siguiente resultado esperado:

```
Algoritmo correccion_Para
Para <-0 Hasta Con Paso Hacer
    Escribir "Imprimimos el valor de i"
    Escribir i
Fin Para
FinAlgoritmo
```

¿Cuál es el resultado a lograr?

PSelnt - Ejecutando proceso SIN_TITULO

```
Imprimimos el valor de i
2
Imprimimos el valor de i
4
Imprimimos el valor de i
6
Imprimimos el valor de i
8
Imprimimos el valor de i
10
Imprimimos el valor de i
12
*** Ejecución Finalizada. ***
```

No cerrar esta ventana Siempre visible Reiniciar ▼

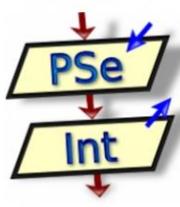


Revisemos lo aprendido hasta aquí

Identificar, construir y utilizar una estructura PARA. En esta estructura tenemos el control absoluto de las repeticiones, ya que podemos determinar la inicialización, límite y aumento del valor de i.

PROGRAMACIÓN DESDE CERO

ESTRUCTURAS DE CONTROL CON PSEINT – ESTRUCTURAS REPETITIVAS



Universidad de
LA PUNTA



GOBIERNO DE
SAN LUIS



EGG



ESTRUCTURAS ANIDADAS

Ya hemos visto todas las estructuras de control posibles, tanto las selectivas y las repetitivas, pero lo que no vimos es que podemos usar ambas estructuras de una manera diferente. En algunas ocasiones, requerimos utilizar estructuras dentro de estructuras. Esto suele ser muy útil como una forma de organizar estructuras de datos de forma más eficiente, para esto vamos a anidar una estructura dentro de otra, vamos a empezar bien el **SI-NO** anidado.

SI-NO ANIDADO

En la guía anterior vimos el uso de la instrucción *Según* para cuando vamos a tener más de dos alternativas para elegir, o una variable que puede tomar varios valores.

Pero, para esto también podemos utilizar la instrucción *Si* para diseñar estructuras de selección que contengan más de dos alternativas. Por ejemplo, una estructura *Si-entonces* puede contener otra estructura *Si-entonces*, y esta estructura *Si-entonces* puede contener otra, y así sucesivamente cualquier número de veces; a su vez, dentro de cada estructura pueden existir diferentes acciones, a esto se le llama condicionales anidados o en cascada.

CONDICIONALES ANIDADOS EN PSEINT:

```
Si expresion_logica1 Entonces
    acciones_por_verdadero1
Sino
Si expresion_logica2 Entonces
    acciones_por_verdadero2
Sino
    Si expresion_logica4 Entonces
        acciones_por_verdadero3
    Sino
        acciones_por_falso
    Fin Si
Fin Si
Fin Si
```



¿NECESITAS UN EJEMPLO?

```
1  Algoritmo SiAnidado
2      Definir nota Como Entero
3      Escribir "Ingrese su nota"
4      Leer nota
5      /// Anidamos los si para tener una accion para las distintas posibilidades
6      Si nota ≤ 6 Entonces
7          Escribir "Desaprobo"
8      SiNo
9          Si nota = 7
10         Escribir "Aprobo"
11     SiNo
12         Si nota = 8
13         Escribir "Muy bien"
14     SiNo
15         Si nota = 9 Entonces
16             Escribir "Sobresaliente"
17         SiNo
18             Si nota = 10
19                 Escribir "Excelente"
20
21             FinSi
22         FinSi
23
24     FinSi
25 Fin Si
26 FinSi
27
28 FinAlgoritmo
```



Pueden encontrar un ejemplo para descargar de Si-No Anidados en el Aula Virtual.

BUCLAS ANIDADOS

Anidar un bucle consiste en meter ese bucle dentro de otro. La anidación de bucles es necesaria para hacer determinados procesamientos.

Un bucle anidado tiene una estructura como la que sigue. Vamos a tratar de explicarlo a la vista de estas líneas:

```
Para i <-1 Hasta 10 Con Paso 1 Hacer
    Para j <-1 Hasta 10 Con Paso 1 Hacer
        Escribir i “-“ j
    Fin Para
Fin Para
```



La ejecución funcionará de la siguiente manera. Para empezar se inicializa el primer bucle, con lo que la variable i valdrá 0 y a continuación se inicializa el segundo bucle, con lo que la variable j valdrá también 0. En cada iteración se imprime el valor de la variable i, un guion ("") y el valor de la variable j, como las dos variables valen 0, se imprimirá el texto "0-0" en la página web.

Debido al flujo del programa en esquemas de anidación como el que hemos visto, el bucle que está anidado (más hacia dentro) es el que más veces se ejecuta. En este ejemplo, para cada iteración del bucle más externo el bucle anidado se ejecutará por completo una vez, es decir, hará sus 10 iteraciones. En la página web se escribirían estos valores, en la primera iteración del bucle externo y desde el principio:

i	j
1	1
1	2
1	3
1	4
1	5
1	6
1	7
1	8
1	9
1	10

Para cada iteración del bucle externo se ejecutarán las 10 iteraciones del bucle interno o anidado. Hemos visto la primera iteración, ahora vamos a ver las siguientes iteraciones del bucle externo. En cada una acumula una unidad en la variable i, con lo que saldrían estos valores.

i	j
2	1
2	2
2	3
2	4
2	5
2	6
2	7
2	8
2	9
2	10

Y luego estos:

i	j
3	1
3	2
3	3
3	4
3	5
3	6
3	7
3	8
3	9
3	10

Así hasta que se terminen los dos bucles, que sería cuando se alcanzase el valor 10-10.



¿NECESITAS UN EJEMPLO?

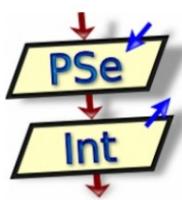
Veamos un ejemplo muy parecido al anterior, aunque un poco más útil. Se trata de imprimir en la página las todas las tablas de multiplicar. Del 1 al 9, es decir, la tabla del 1, la del 2, del 3...

```
1 Algoritmo tablas
2
3     Definir i, j Como Entero
4
5     Para i<-1 Hasta 9 Con Paso 1 Hacer
6
7         Escribir "La tabla del " i ":" 
8
9         Para j<=1 Hasta 9 Con Paso 1 Hacer
10
11             Escribir Sin Saltar i "x" j ":" 
12
13             Escribir (i * j)
14
15             Escribir " "
16
17         Fin Para
18
19     Fin Para
20
21 FinAlgoritmo
22
```

Con el primer bucle controlamos la tabla actual y con el segundo bucle la desarrollamos. En el primer bucle escribimos una cabecera, indicando la tabla que estamos escribiendo, primero la del 1 y luego las demás en orden ascendente hasta el 9. Con el segundo bucle escribo cada uno de los valores de cada tabla.

PROGRAMACIÓN DESDE CERO

SUBPROGRAMAS EN PSEINT





Objetivos de la Guía

En esta guía aprenderemos a:

- Separar el Algoritmo principal de las Funciones y SubProgramas.
- Diferenciar una función de un subprograma.
- Comprender qué debe ejecutarse en una función o subprograma.
- Lograr enviar información a las funciones o subprogramas a través de parámetros por valor o por referencia.
- Diferenciar pasaje por valor y por referencia.
- Llamar funciones o subprogramas desde el Algoritmo Principal.
- Definir variables de retorno y operar con ellas.
- Utilizar estructuras de control en Funciones y Subprogramas.

SUBPROGRAMAS

En esta guía aprenderemos el uso de los subprogramas. Hasta el momento, desarrollamos nuestro código en el Algoritmo Principal, con esto, si quisieramos realizar varias sumas de variables, cada vez que quisieramos adicionar su valor, deberíamos calcularlo, algo así:

```
1  Algoritmo sin_titulo
2
3      Definir num1, num2, num3, num4, num5 Como Entero
4      Definir resultado1, resultado2, resultado3, resultado4 Como Entero
5
6      resultado1 = num1 + num2
7      Escribir resultado1
8
9      resultado2 = num1 + num3
10     Escribir resultado2
11
12     resultado3 = num1 + num4
13     Escribir resultado3
14
15     resultado4 = num1 + num5
16     Escribir resultado4
17
18 FinAlgoritmo
```



Muchas veces nos ocurrirá en la programación que deberemos realizar la misma operación varias, incluso miles de veces. Para evitar que repitamos estas operaciones, surgen los subprogramas. Sería mucho más eficiente hacer lo siguiente:

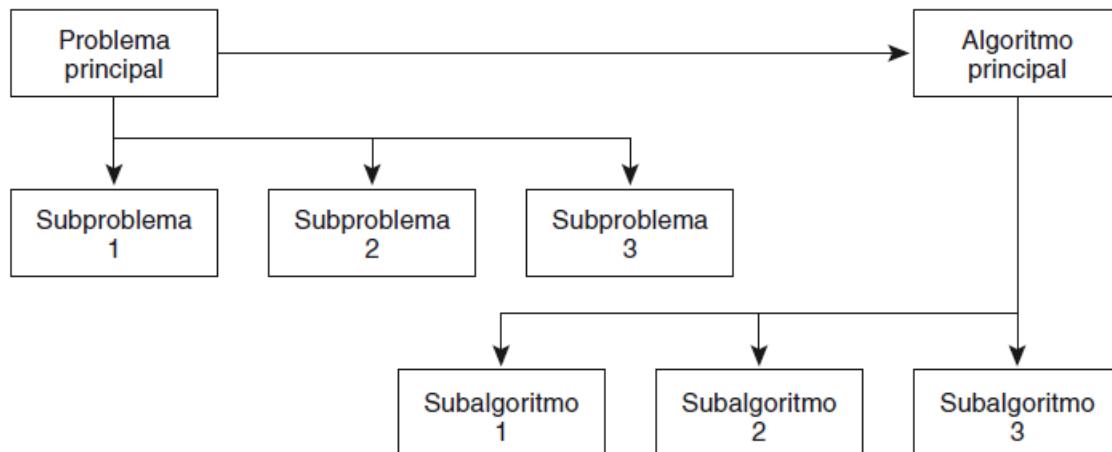
```
Algoritmo sin_titulo
1
2     Definir num1, num2, num3, num4, num5 Como Entero
3
4
5     sumar(num1, num2)
6     sumar(num1, num3)
7     sumar(num1, num4)
8     sumar(num1, num5)
9
10 FinAlgoritmo
11
12
13 SubProceso sumar(x, z)
14     Escribir x+z
15 FinSubProceso
```

Nótese aquí dos cosas importantes: la cantidad de líneas es menor y es mucho más legible y claro el código y su intención.

Un método muy útil para solucionar un problema complejo es dividirlo en **subproblemas** — problemas más sencillos— y a continuación dividir estos subproblemas en otros más simples, hasta que los problemas más pequeños sean fáciles de resolver. Esta técnica de dividir el problema principal en subproblemas se suele denominar “**divide y vencerás**”.

Este método de diseñar la solución de un problema principal obteniendo las soluciones de sus subproblemas se conoce como diseño descendente (top-down). Se denomina descendente, ya que se inicia en la parte superior con un problema general y se termina con varios subproblemas de ese problema general y los soluciones a esos subproblemas. Luego, las partes en que se divide un programa deben poder desarrollarse independientemente entre sí.

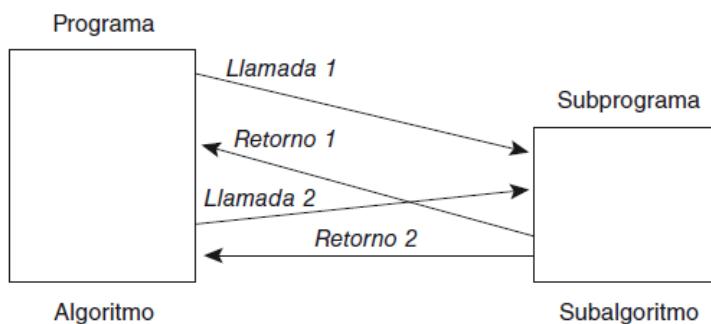
Las soluciones de un diseño descendente pueden implementarse fácilmente en lenguajes de programación y se los denomina subprogramas o sub-algoritmos si se emplean desde el concepto algorítmico.



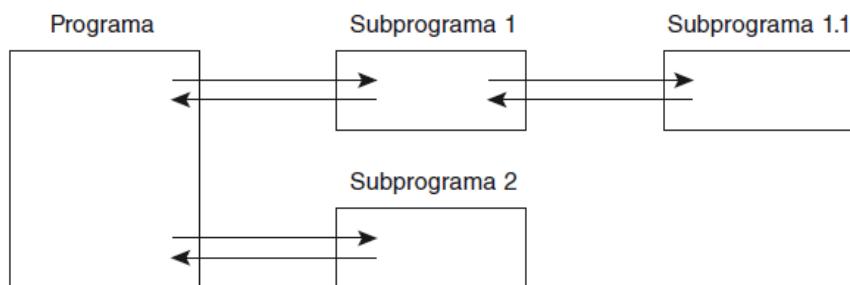
El problema principal se soluciona por el correspondiente programa o algoritmo principal, mientras que la solución de los subproblemas será a través de subprogramas, divididos en **procedimientos y funciones**. Un subprograma es un como un **mini algoritmo**, que recibe los *datos*, necesarios para realizar una tarea, desde el programa y devuelve *resultados* o realiza esa tarea.



Cada vez que el subprograma es invocado, el algoritmo va al subprograma invocado y se realiza el subprograma, y a su vez, un subprograma puede llamar a otros subprogramas.



Un programa con un subprograma: función y procedimiento



Un programa con diferentes niveles de subprogramas

ÁMBITO: VARIABLES LOCALES Y GLOBALES

Las variables utilizadas en los programas principales y subprogramas se clasifican en dos tipos: *variables locales* y *variables globales*.

Una **variable local** es aquella que está declarada y definida dentro de un subprograma, en el sentido de que está dentro de ese subprograma, es distinta de las variables con el mismo nombre declaradas en cualquier parte del programa principal y son variables a las que el algoritmo principal no puede acceder de manera directa.

El significado de una variable se confina al procedimiento en el que está declarada. Cuando otro subprograma utiliza el mismo nombre se refiere a una posición diferente en memoria. Se dice que tales variables son locales al subprograma en el que están declaradas.

Una **variable global** es aquella que está declarada en el programa o algoritmo principal, del que dependen todos los subprogramas y a las que pueden acceder los subprogramas, a través del paso de argumento. La parte del programa/algoritmo en que una variable se define se conoce como **ámbito** o **alcance** (scope, en inglés).

El uso de variables locales tiene muchas ventajas. En particular, hace a los subprogramas independientes, siendo solo la comunicación entre el programa principal y los subprogramas a través de la lista de parámetros.

Una variable local a un subprograma no tiene ningún significado en otros subprogramas. Si un subprograma asigna un valor a una de sus variables locales, este valor no es accesible a otros programas, es decir, no pueden utilizar este valor. A veces, también es necesario que una variable tenga el mismo nombre en diferentes subprogramas. Por el contrario, las variables globales tienen la ventaja de compartir información de diferentes subprogramas sin la necesidad de ser pasados como argumento.

COMUNICACIÓN CON SUBPROGRAMAS: PASO DE ARGUMENTOS

Cuando un programa llama a un subprograma, la información se comunica a través de la lista de parámetros y se establece una correspondencia automática entre los parámetros y los argumentos. Los parámetros son “sustituidos” o “utilizados” en lugar de los argumentos.

La declaración del subprograma se hace con:

```
Subproceso nombre (PA1, PA2, ..., PAn)
    <acciones>
```

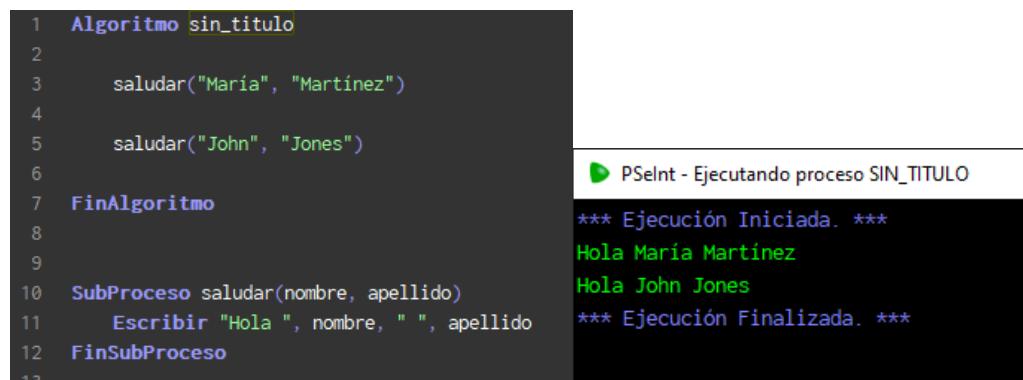
```
FinSubproceso
```

y la llamada al subprograma con:

```
nombre (AR1, ARG2,..., ARGn)
```

donde **PA1, PA2, ..., PAn** son los parámetros y **ARG1, ARG2, ..., ARGn** son los argumentos.

Para dar un ejemplo de la vida real, tenemos que ver a los subprogramas como formularios vacíos que esperan ser llenados. Es decir, si yo tengo un formulario que espera que complete ‘nombre’ y ‘apellido’ esos serían los parámetros del subprograma. Al llenar con mis datos, estoy dando mi nombre y apellido como argumento. Esto permite que muchas personas puedan llenar el mismo formulario, pero cada uno con su información. Miremos el siguiente ejemplo donde la función tiene los parámetros genéricos nombre y apellido, pero al llamarlo desde el algoritmo le pasamos nombres concretos.



```
1  Algoritmo sin_titulo
2
3      saludar("Maria", "Martinez")
4
5      saludar("John", "Jones")
6
7  FinAlgoritmo
8
9
10 SubProceso saludar(nombre, apellido)
11     Escribir "Hola ", nombre, " ", apellido
12 FinSubProceso
```

PSelnt - Ejecutando proceso SIN_TITULO
*** Ejecución Iniciada. ***
Hola Maria Martinez
Hola John Jones
*** Ejecución Finalizada. ***

Cuando nosotros decidimos los parámetros que va a necesitar nuestro subprograma, también podemos decidir cuál va a ser el comportamiento de los argumentos en nuestro subprograma cuando lo invoquemos y se los pasemos por paréntesis. Esto va a afectar directamente a los argumentos y no al resultado final del subprograma.

Para esto existen dos tipos más empleados para realizar el paso de argumentos, el **paso por valor** y el **paso por referencia**.

PASO POR VALOR

Los argumentos se tratan como variables locales y los valores de dichos argumentos se proporcionan copiando los valores de los argumentos originales. Los parámetros (locales a la



función o procedimiento) reciben como valores iniciales una copia de los valores de los argumentos y con ello se ejecutan las acciones descritas en el subprograma.

Aunque el paso por valor es sencillo, tiene una limitación acusada: no existe ninguna otra conexión con los parámetros, y entonces los cambios que se produzcan dentro del subprograma no producen cambios en los argumentos originales y, por consiguiente, no se pueden poner argumentos como valores de retorno. El argumento actual no puede modificarse por el subprograma.

En PSeInt todas las variables que pasemos como argumentos pasan por defecto “**Por Valor**” sino se especifica lo contrario explícitamente.

PASO POR REFERENCIA

En numerosas ocasiones se requiere que ciertos argumentos sirvan como argumentos de salida, es decir, se devuelvan los resultados al programa que llama. Este método se denomina **paso por referencia** o también de llamada por dirección o variable. El programa que llama pasa al subprograma la dirección del argumento actual (que está en el programa que llama). Una referencia al correspondiente argumento se trata como una referencia a la posición de memoria, cuya dirección se ha pasado. Entonces una variable pasada como argumento real es compartida, es decir, se puede modificar directamente por el subprograma.

La característica de este método se debe a su simplicidad y su analogía directa con la idea de que las variables tienen una posición de memoria asignada desde la cual se pueden obtener o actualizar sus valores. El área de almacenamiento (direcciones de memoria) se utiliza para pasar información de entrada y/o salida; en ambas direcciones.

En este método los argumentos son de entrada/salida y los argumentos se denominan **argumentos variables**.



MANOS A LA OBRA!

Copia, pega y ejecuta el código. Analiza qué está sucediendo

Algoritmo valorVSreferencia

Definir num Como Entero

num = 2

Escribir num

Escribir "Ahora enviamos el número a la función por valor y el resultado es:"

elevarAlCuadradoPorValor(num)

Escribir num

Escribir "*****"

Escribir "Ahora enviamos el número a la función por referencia y el resultado es:"

elevarAlCuadradoPorReferencia(num)

Escribir num

FinAlgoritmo



SubProceso elevarAlCuadradoPorValor(num Por Valor)

num = num * num

FinSubProceso

SubProceso elevarAlCuadradoPorReferencia(num Por Referencia)

num = num * num

FinSubProceso



Revisemos lo aprendido hasta aquí

- Variables globales y locales
- Pasaje de datos por valor y por referencia

¿QUÉ SON LAS FUNCIONES?

Matemáticamente una función es una operación que toma uno o más valores llamados argumentos y produce un resultado.

Cada función se evoca utilizando su nombre en una expresión con los argumentos encerrados entre paréntesis. A una función no se le llama explícitamente, sino que se le invoca o referencia mediante un nombre y una lista de parámetros.

¿CÓMO SE DECLARA UNA FUNCIÓN?

Cada función se crea fuera de nuestro algoritmo y requiere de una serie de pasos que la definen. Una función como tal subalgoritmo o subprograma tiene una constitución similar a los algoritmos. Esta comenzará con la palabra reservada **Función** y termina con la palabra **FinFunción** al igual que un Algoritmo. Al lado de nuestra palabra **Función**, comenzaremos la creación de nuestra función, esta constará de una cabecera que comenzará con un nombre para el **valor devuelto por la función**. El valor devuelto será una variable que definiremos dentro del cuerpo de nuestra función, ahí la daremos un tipo de dato. Este valor devuelto debe ser el resultado de la tarea que hemos dividido del problema general.

Después, va a ir el nombre de nuestra función y a continuación, entre paréntesis los parámetros de dicha función. Los parámetros van a ser los datos que necesitamos que nos envíe el algoritmo para realizar el subproblema en cuestión. Estos se escriben poniendo el nombre de la variable a recibir, sin su tipo de dato, y si quisieramos pasar más de una variable, los sepáramos con comas. Los parámetros no son obligatorios a la hora de usar un subprograma, podemos tener una función sin parámetros, aunque es poco común.

Por último, irá el cuerpo de la función, que será una serie de acciones o instrucciones cuya ejecución hará que se asigne un valor al nombre de la función. Esto determina el valor particular del resultado que ha de devolverse al programa llamador.

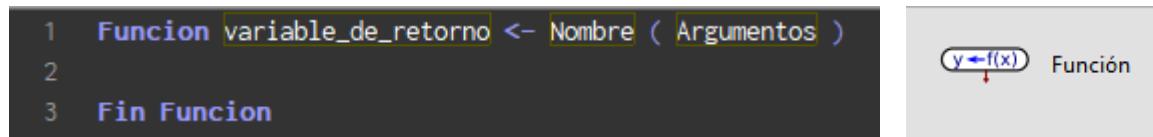


El algoritmo o programa llama o invoca a la función con el nombre de esta última en una expresión seguida de una lista de argumentos que deben coincidir en cantidad, tipo y orden con los parámetros de la función. Se denominan argumentos a las variables o valores declarados en el algoritmo. Cuando se realiza una llamada a la función, los "valores" pasados o enviados a la función se denominan argumentos.

SINTAXIS

```
Funcion variable_de_retorno <- Nombre (Parámetros)
Definir variable_de_retorno como Tipo de Dato
<acciones> //cuerpo de la función
FinFuncion
```

¿Cómo se ve en PseInt?

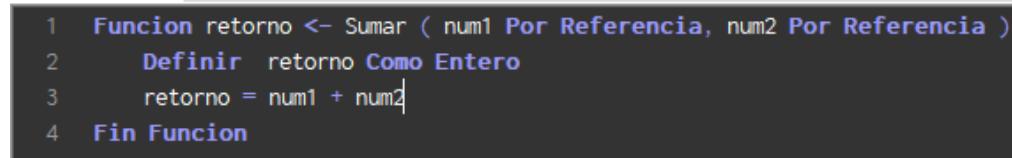


```
1 Funcion variable_de_retorno <- Nombre ( Argumentos )
2
3 Fin Funcion
```

- **Parámetros:** uno o más parámetros de la siguiente forma: (parámetro 1 [Por Valor/Por Referencia], parámetro 2 [Por Valor/Por Referencia],...]).
- **Nombre asociado con la función:** que será un nombre de identificador válido.
- **<acciones>:** instrucciones que constituyen la definición de la función y que debe contener alguna instrucción mediante la cual se asigne un valor a la variable_de_retorno.
- **Variable_de_retorno:** Esta variable la escribiremos en la definición de la función y debemos Definir su tipo dentro de la Función para poder usarla. En ella alojaremos el resultado final de la función. Cuando llamemos a la función, con los debidos argumentos, el algoritmo principal recibirá el valor de esta variable de retorno.



¿NECESITAS UN EJEMPLO?



```
1 Funcion retorno <- Sumar ( num1 Por Referencia, num2 Por Referencia )
2     Definir retorno Como Entero
3     retorno = num1 + num2
4 Fin Funcion
```



Para crear una función utiliza la plantilla que encuentras en el panel desplegable a la derecha de la pantalla.



EJERCICIO COOPERAR

Realiza una función llamada Cooperar que reciba dos variables de tipo carácter, una variable debe contener el mensaje “Cooperando” y la otra “trabajamos mejor”. La función debe concatenar ambos textos.

DETECCIÓN DE ERRORES

¿Puedes corregir esta función para que cumpla con su sintaxis?

```
Func retorno <- Paridad ( num
retorno : num MOD 2 == 0
Fin Funcion
```



Revisemos lo aprendido hasta aquí

- Seleccionar la plantilla de funciones desde el panel lateral.
- Nombrar funciones.
- Asignar parámetros a las funciones.
- Identificar, definir y operar con el retorno.

¿CÓMO SE INVOCAN LAS FUNCIONES?

Una función puede ser llamada de la siguiente forma:

```
nombre_función(Argumentos)
```

- *nombre_función*: función que va a llamar.
- *argumentos*: constantes, variables, expresiones.

Cada vez que se llama a una función desde el algoritmo principal se establece automáticamente una correspondencia entre los argumentos y los parámetros. Debe haber exactamente el mismo número de parámetros que de argumentos en la declaración de la función y se presupone una correspondencia uno a uno de izquierda a derecha entre los argumentos y los parámetros.

Además, cuando se llama a la función está va a devolver el resultado de las acciones realizadas en la función(**variable de retorno**) este resultado debe ser “atrapado” en el algoritmo. Ya sea para usarlo o solo para mostrarlo, por lo que al llamar una función debemos, o asignarle el resultado a una variable o concatenar el llamado de una función con un escribir

```
variable = nombre_funcion(argumentos)
```



Escribir nombre_funcion(argumentos)

Una llamada a la función implica los siguientes pasos:

1. A cada argumento se le asigna el valor real de su correspondiente parámetro.
2. Se ejecuta el cuerpo de acciones de la función.
3. Se devuelve el valor de la función y se retorna al punto de llamada.

Entonces para resumir se puede decir que la función tiene cinco componentes importantes:

- el **identificador**: va a ser el nombre de la función, mediante el cual la invocaremos.
- los **parámetros** son los valores que recibe la función para realizar una tarea.
- los **argumentos**, son los valores que envía el algoritmo a la función.
- las **acciones de la función**, son las operaciones que hace la función.
- **valor de retorno(o el resultado)** , es el valor final que entrega la función. La función devuelve un **único valor**.



¿NECESITAS UN EJEMPLO?

```
1 Funcion retorno <- Sumar ( num1 Por Referencia, num2 Por Referencia )
2   Definir retorno Como Entero
3   retorno = num1 + num2
4 Fin Funcion
5
6 Algoritmo sin_titulo
7   Definir num1, num2, num3, num4, num5, num6, resultado Como Entero
8   resultado = Sumar(num1, num2)
9   Escribir Sumar(num3, num4)
10 FinAlgoritmo
11
```



MANOS A LA OBRA!

EJERCICIO COOPERAR – PARTE 2

¿Recuerdan la Función Cooperar? Hora de llamarla en el algoritmo principal y mostrar el mensaje por pantalla.



Una buena costumbre es alojar el retorno de las funciones en variables, ya que esto guardará el resultado por si debemos operar con él más adelante.

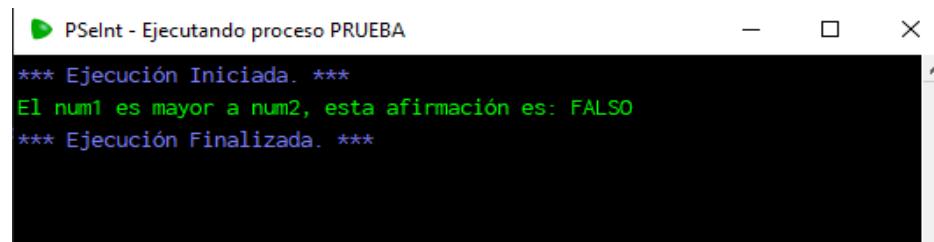


DETECCIÓN DE ERRORES

¿Puedes corregir esta función para lograr el resultado esperado?

```
Funcion retorno <- Comparar ( num1 )
retorno = num1 num2
Algoritmo Prueba
Definir num1, num2 Como Entero
Definir resultado Como Logico
num1 = 3
num2 = 6
resultado = retorno(num1,num2)
Escribir "El num1 es mayor a num2, esta afirmación es: " resultado
FinAlgoritmo
```

¿Cuál es el resultado a lograr?



```
PSelint - Ejecutando proceso PRUEBA
*** Ejecución Iniciada. ***
El num1 es mayor a num2, esta afirmación es: FALSO
*** Ejecución Finalizada. ***
```



Pueden encontrar ejemplos para descargar de Funciones en Aula Virtual.

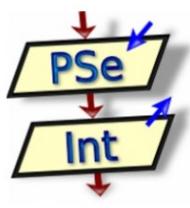


Revisemos lo aprendido hasta aquí

- Declarar funciones
- Invocar a las funciones desde el algoritmo principal
- Enviar argumentos al invocar.
- Revisar la Función Cooperar de un compañero y compararla con la propia.

PROGRAMACIÓN DESDE CERO

SUBPROGRAMAS EN PSEINT



Universidad de
LA PUNTA





Objetivos de la Guía

En esta guía aprenderemos a:

- Separar el Algoritmo principal de las Funciones y SubProgramas.
- Diferenciar una función de un subprograma.
- Comprender qué debe ejecutarse en una función o subprograma.
- Lograr enviar información a las funciones o subprogramas a través de parámetros por valor o por referencia.
- Diferenciar pasaje por valor y por referencia.
- Llamar funciones o subprogramas desde el Algoritmo Principal.
- Definir variables de retorno y operar con ellas.
- Utilizar estructuras de control en Funciones y Subprogramas.

PROCEDIMIENTOS

Ahora que sabemos que las **funciones devuelven un valor** para que lo usemos a través de la variable de retorno, vamos a ver un caso distinto. Los **procedimientos** son similares, pero **no devuelven ningún valor**, sólo realizan una tarea.

¿QUÉ ES UN PROCEDIMIENTO?

Un **procedimiento** es un subprograma que ejecuta un proceso específico. En PseInt lo llamaremos **SubProceso**. Ningún valor está asociado con el nombre del procedimiento; por consiguiente, no puede ocurrir en una expresión. Un procedimiento se llama escribiendo su nombre. Cuando se invoca el procedimiento, los pasos que lo definen se ejecutan y a continuación se devuelve el control al programa que le llamó.

Sintaxis

SubProceso Nombre (*parámetros*)

<*acciones*>

FinSubProceso

Los parámetros tienen el mismo significado que en las funciones.

¿CÓMO INVOCAMOS A UN SUBPROGRAMA?

Un procedimiento puede ser llamado de la siguiente forma:

nombre(argumentos)

- nombre_procedimiento: procedimiento que se va a llamar.
- argumentos: constantes, variables, expresiones.



Pueden encontrar ejemplos para descargar de Procedimientos en Aula Virtual.

RECUSIÓN

Una función o procedimiento que se puede llamar a sí mismo se llama recursivo. La recursión (recursividad) es una herramienta muy potente en algunas aplicaciones, sobre todo de cálculo. La recursión puede ser utilizada como una alternativa a la repetición o estructura repetitiva. El uso de la recursión es particularmente idóneo para la solución de aquellos problemas que pueden definirse de modo natural en términos recursivos.

La escritura de un procedimiento o función recursiva es similar a sus homónimos no recursivos; sin embargo, para evitar que la recursión continúe indefinidamente es preciso incluir una condición de terminación, que suele hacerse con una estructura condicional.

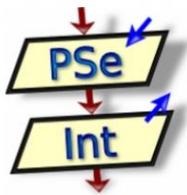


Revisemos lo aprendido hasta aquí

- Declarar subprocessos
- Invocar a subprocessos desde el algoritmo principal
- Enviar argumentos al invocar.
- Qué es la recursión y para qué sirve

PROGRAMACIÓN DESDE CERO

EJERCICIO COOPERATIVO GUÍA 3



Ejercicio Cooperativo

¿Qué es?

Este ejercicio debe realizarse con todos los integrantes del equipo aportando su opinión y visión de resolución. Son ejercicios de los que **vas a aprender mucho, no por el ejercicio en sí, sino por ver cómo tus compañeros piensan y resuelven.**

Si aún no has terminado la guía, ¡no te preocupes! Realizar un ejercicio con tus compañeros te ayudará a revisar los conceptos y ponerlos en práctica, luego puedes continuar con tus ejercicios.

Este ejercicio debe empezarse y terminarse el día que se habilita y el tiempo que deben dedicarle sería entre 30 y 45 minutos. Puedes pautar con tus compañeros un determinado horario para hacerlo.

Si eres **MENTOR**, mientras esperas que el Coach te asigne una mesa para colaborar, puedes repasar lo aprendido en esta guía haciendo este ejercicio de manera personal.

¿Qué sucede si no terminamos?

Aunque no hayan logrado llegar al resultado final, el trabajo en equipo para su desarrollo, pensamiento y lógica del mismo les ayudará a afianzar los conocimientos vistos, a que pongan en práctica una vez más la cooperación y se enriquezcan académicamente de las opiniones y visiones de los miembros del equipo.

Ejercicio

Vamos a programar una calculadora de materiales para construir

Primero leeremos todo el ejercicio y luego **dividiremos tareas en el equipo.**

El algoritmo principal sólo debe llamar al subPrograma menu()

Cada subPrograma puede descomponerse, si hiciera falta, en otros subProgramas a creatividad del programador

El menú debe quedar de la siguiente manera:

- 1 - Calcular muro de ladrillo
- 2 - Calcular viga de hormigón
- 3 - Calcular columnas de hormigón
- 4 - Calcular contrapisos
- 5 - Calcular techo
- 6 - Calcular pisos
- 7 - Calcular pintura
- 8 - Calcular iluminación
- 9 - Salir

subprogramas calcularSuperficie y calcularVolumen

Haremos ambos para usarlos dentro de los otros subprogramas. El usuario no puede acceder a ellos.

Ejercicio Cooperativo

subprograma calcularMuro

Nos debe pedir primero si el muro será de 20 o 30 cm de espesor. Luego el largo y el alto. A partir de estos datos se debe mostrar al usuario la superficie del muro y la cantidad de materiales que necesitaremos para construirlo.

Si el muro es de 30cm necesitaremos por metro cuadrado: 15.2 kg de cemento, 0.115 m³ de arena y 120 ladrillos.

Si el muro es de 20cm necesitaremos por metro cuadrado: 10.9 kg de cemento, 0.09 m³ de arena y 90 ladrillos.

subprograma calcularViga

Nos debe pedir el largo de la viga. Por metro lineal de viga se necesitarán: 9 kg de cemento, 0.02 m³ de arena, 0.02 m³ de piedra, 4 m de hierro del 8 y 3 m de hierro del 4.

Debemos mostrar al usuario la cantidad de materiales necesaria.

subprograma calcularColumna

Nos debe pedir el largo de la columna. Por metro lineal de columna se necesitarán: 7.5 kg de cemento, 0.016 m³ de arena, 0.016 m³ de piedra, 6 m de hierro del 10 y 3 m de hierro del 4.

Debemos mostrar al usuario la cantidad de materiales necesaria.

subprograma calcularContrapisos

Nos debe pedir espesor, ancho y largo del contrapiso a calcular.

Por metro cúbico de contrapiso se necesita: 105 kg de cemento, 0.45 m³ de arena y 0.9 m³ de piedra.

Debemos mostrar al usuario la cantidad de materiales necesaria.

subprograma calcularTecho

Nos debe pedir espesor, ancho y largo del techo a calcular.

Por metro cúbico de techo se necesita: 33 kg de cemento, 0.072 m³ de arena, 0.072 m³ de piedra, 7 m de hierro del 8 y 4 m de hierro del 6

Debemos mostrar al usuario la cantidad de materiales necesaria.

subprograma calcularPisos

Nos debe pedir ancho y largo del paño de piso a colocar. Teniendo esos datos se debe calcular la superficie y añadirle un 10% extra por recortes

Mostrar el resultado en m²

subprograma calcularPintura

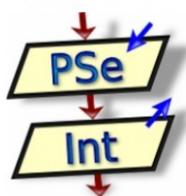
Nos debe pedir la superficie del muro y mostrar cuánta pintura necesitamos teniendo en cuenta que rinde 6 m² por litro de pintura.

subprograma calcularIluminacion

Nos debe pedir la superficie de la habitación. La iluminación la calculamos de la siguiente forma: superficie * 0.20. Eso nos da la cantidad mínima de superficie de iluminación natural (ventanas y puertas de vidrio). Mostrar resultado

PROGRAMACIÓN DESDECERO

ARREGLOS CON PSEINT



Universidad de
LA PUNTA





Objetivos de la Guía

En esta guía aprenderemos a:

- Definir arreglos de acuerdo al tipo de dato que contendrán.
- Dimensionar arreglos.
- Rellenar arreglos.
- Mostrar correctamente arreglos por pantalla.
- Utilizar funciones y subprogramas, y estructuras de control para trabajar con arreglos y matrices.

¿QUÉ SON LOS ARREGLOS?

En guías previas la manera de manipular datos era a través de variables, las variables nos dejaban manejar de a un dato a la vez, pero si necesitáramos manejar varios datos juntos en un mismo lugar, usaríamos los arreglos. Si vieramos las variables como cajas, estas nos permitían guardar un solo dato. Los arreglos son como cajas, pero dentro hay compartimentos, que permiten guardar varios datos, siempre y cuando sean del mismo tipo.

Un array o arreglo (matriz o vector) es un conjunto finito y ordenado de elementos homogéneos. La propiedad “ordenado” significa que el elemento primero, segundo, tercero, ..., enésimo de un arreglo puede ser identificado. Los elementos de un arreglo son homogéneos, es decir, del mismo tipo de datos. Un arreglo puede estar compuesto de todos sus elementos de tipo cadena, otro puede tener todos sus elementos de tipo entero, etc, pero no puede ser de datos distintos. Los arreglos también pueden utilizarse en expresiones lógicas si necesitásemos comprobar varios elementos a la vez, o si necesitásemos saber si un elemento de nuestro arreglo, existe dentro del arreglo.

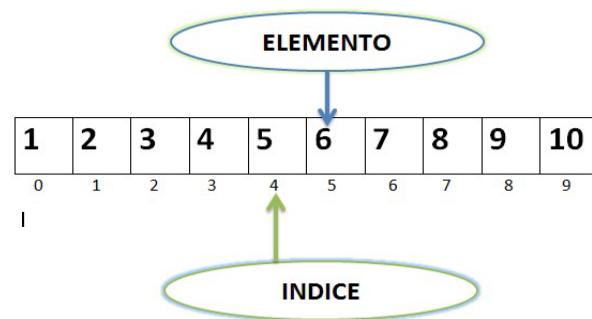
ARREGLOS UNIDIMENSIONALES: VECTORES

¿QUÉ SON LOS VECTORES?

El tipo más simple de arreglo es el arreglo unidimensional o vector. Un vector es un arreglo de n elementos, uno detrás de otro, que posee las siguientes características:

- Se identifica por un único nombre de variable.
- Sus elementos se almacenan en posiciones del vector y cada a posición le corresponde un subíndice.
- Se puede acceder a cada uno de sus elementos a través del subíndice de forma ordenada o en forma aleatoria. □ Su tamaño es finito, esto significa que una vez definido su tamaño, este no

puede cambiar. El tamaño es la cantidad de elementos que puede guardar nuestro vector.



Matemáticamente se ve algo así: $v = [1, 2, 3, 4 \dots n]$

Subíndice

- El subíndice es el número entero que identifica cada elemento dentro del vector, sin importar el tipo de dato que posea.
- Un vector de tamaño N posee N subíndices que se suceden de forma creciente y monótona.
Ejemplo: 0 – 1 – 2 – 3 – 4 – 5 – 6 – N
- El valor inicial del primer subíndice depende del lenguaje; la mayoría de los modernos inician con el cero, por lo tanto, en PSeInt comenzarán en cero y los posibles valores de los subíndices irán desde 0 hasta N-1.

Declaración

```
Definir nombre_vector como Tipo_de_Dato
```

```
Dimension nombre_vector(tamaño)
```

Donde Tipo_De_Dato se corresponde con cualquiera de los tipos de datos simples vistos previamente: entero, real, cadena, lógico.

La declaración Dimension nos sirve para darle el tamaño a nuestro vector, que recordemos, no puede cambiar una vez declarado. El tamaño va a ser siempre un numero entero o una variable entera, el tamaño no puede ser un numero con decimales.

El tamaño nos sirve para declarar cuantos elementos va a poder guardar nuestro vector. Si decimos que nuestro vector va a guardar 5 elementos, no puede guardar 6 o nos producirá un error.

Algoritmo vectores

```
Definir vector Como Entero
```

```
Dimension vector[5]
```

```
FinAlgoritmo
```



MANOS A LA OBRA!

EJERCICIO DEFINIR VECTOR

Define un vector que alojará números enteros y otro de cadena. Dimensiona ambos de la longitud que tu deseas. Ahora en lapiz y papel, escribe la dimensión de tus vectores y sus subíndices.

ASIGNACIÓN DE VALORES

Cuando queremos ingresar un elemento en nuestro arreglo vamos a tener que elegir el subíndice en el que lo queremos guardar. Una vez que tenemos el subíndice decidido tenemos que invocar nuestro vector por su nombre y entre paréntesis el subíndice en el que lo queremos guardar. Después, pondremos el signo de igual (que es el operador de asignación) seguido del elemento a guardar.

El elemento a guardar debe coincidir con el tipo de dato de nuestro arreglo, si nuestro arreglo es de tipo entero, solo podemos guardar números enteros. También sucede algo parecido con el subíndice, no podemos llamar un subíndice que no existe, recordemos que los subíndices dependen del tamaño de nuestro arreglo. Entonces si tenemos un arreglo de tamaño 5, no podemos llamar el subíndice 6 porque no existe.

Algoritmo vectores

Definir vector Como Entero

Dimension vector[5]

```
vector[0] = 1
vector[1] = 2
vector[2] = 3
vector[3] = 4
vector[4] = 5
```

FinAlgoritmo

Esta forma de asignación implica asignar todos los valores de nuestro arreglo de uno en uno, esto va a conllevar un trabajo bastante grande dependiendo del tamaño de nuestro arreglo.

Entonces, para poder asignar varios valores a nuestro arreglo y no hacerlo de uno en uno usamos un bucle Para. El bucle Para, al poder asignarle un valor inicial y un valor final a una variable, podemos adaptarlo fácilmente a nuestros arreglos. Ya que, pondríamos el valor inicial de nuestro arreglo y su valor final en las respectivas partes del Para. Nosotros, usaríamos la variable creada en el Para, y la pasaríamos a nuestro arreglo para representar todos los subíndices del arreglo, de esa manera, recorriendo todas las posiciones de nuestro arreglo, asignándole a cada posición un elemento.



Algoritmo vectores

```
Definir vector,i Como Entero
Dimension vector[5]
Para i<-0 Hasta 4 Con Paso 1 Hacer
    vector[i] = i
Fin Para
FinAlgoritmo
```

Nuestra variable *i* pasara por todos los subíndices de nuestro arreglo, ya que ira desde 0 hasta 5. Recordemos que los arreglos arrancan de 0, entonces, debemos calcular que, si el tamaño que le definimos al arreglo es de 5, necesitamos que nuestro *Para* vaya de 0 a 4



MANOS A LA OBRA!

EJERCICIO LLENAR VECTOR

Ahora es tu turno. Llena uno de los vectores que definiste y dimensionaste anteriormente, de forma manual y el otro con un Bucle *Para*.

DETECCIÓN DE ERRORES

Copia y pega el código que está a continuación, tu tarea es arreglar el código.

```
Algoritmo vectores Definir
vector Como Entero
Dimensionvector()
Para i<-0 Hasta 5 Con Paso 1 Hacer
vector(0)=0 Fin Para
FinAlgoritmo
```

MOSTRAR O TRAER ELEMENTOS DE UN ARREGLO

A la hora de querer mostrar o traer algún elemento de nuestro arreglo, lo único que tenemos que hacer es escribir el nombre de nuestro arreglo y entre llaves o paréntesis pasarle un subíndice de ese arreglo para que traiga el elemento que se encuentra en ese subíndice.

```
Definir vector, var Como Entero  
  
Dimension vector[5]  
  
Escribir vector[2]  
  
var = vector[3]
```

Si quisieramos mostrar todos los elementos de nuestro arreglo, deberíamos usar una estructura Para, que recorrerá todos los subíndices de nuestro arreglo y así poder mostrarlos todos.

```
Para i<-0 Hasta 4 Con Paso 1 Hacer  
  
    Escribir Sin Saltar "[" vector[i] "]"  
  
Fin Para
```

Nuestra variable i pasara por todos los subíndices de nuestro arreglo, ya que ira desde 0 hasta 4. Esto es porque como los arreglos arrancan de 0, debemos calcular que, si el tamaño que le definimos al arreglo es de 5, necesitamos que nuestro Para vaya de 0 a 4.



Utilizamos Escribir Sin Saltar para que el programa no muestre saltos de línea al imprimir el vector.



MANOS A LA OBRA!

EJERCICIO MOSTRAR VECTOR

Ahora te toca a ti mostrar tus vectores. Además, define una nueva variable y aloja allí algún valor del vector.

USO EN SUBPROGRAMAS

Los arreglos, cualquiera sea su dimensión, se pueden pasar como parámetros a un subprograma (función o procedimiento) del mismo modo que las variables escalares. Sin embargo, hay que tener en cuenta que los arreglos, a diferencia de los tipos de datos simples, pasan siempre como parámetro “Por Referencia”, ya que usualmente en nuestros subprogramas usamos los arreglos para llenar, mostrar nuestros arreglos, etc.

```
Funcionvariable_de_retorno<- Nombre (vector por referencia)  
  
Definir variable_de_retorno como Tipo de Dato  
  
<acciones>  
  
Fin Funcion
```

```
SubProceso Nombre (matriz por referencia)  
<acciones>
```

FinSubProceso

¿Cómo se ve en PseInt?

```
SubProceso ejemploSubProceso ( vector Por Referencia )
```

```
//Acciones
```

```
FinSubProceso
```

```
Algoritmo vectores
```

```
    Definir vector Como Entero
```

```
    Dimension vector[5]
```

```
FinAlgoritmo
```