

“La modularización no trata de escribir más código, sino de pensar mejor el código: dividir para comprender, reutilizar y construir programas más claros.”

¿Qué es la Modularización?

La modularización es una forma de organizar y resolver un problema complejo dividiéndolo en partes más simples y manejables, llamadas subalgoritmos o módulos.

Cada subalgoritmo cumple una tarea específica, bien definida y autocontenida. Esto significa que:

- Hace solo una cosa, una tarea concreta y claramente definida.
- Puede usarse varias veces, en un mismo programa o en distintos programas.
- No depende directamente de otros módulos, sino que se comunica mediante parámetros.

En PSeInt, la modularización se implementa por subalgoritmos, que pueden ser de dos tipos:

- **Procedimientos**, cuando principalmente procesan datos pero *no devuelven un resultado* directo.
- **Funciones**, cuando realizan algún cálculo que se **retorna** en un valor que puede guardarse en una variable o usarse dentro del programa. Aquí usamos variables de retorno. (leer el manual)

http://servicios.dirinfo.unsl.edu.ar/abm/assets/uploads/materiales/b1e28-2025-pseint_subalgoritmos.pdf

Una síntesis de lo más importante que deben comprender

1. Ahora trabajamos con un **Algoritmo** y uno o más **subalgoritmos** para resolver un problema.
2. Un **subalgoritmo** es un bloque que resuelve una tarea específica. Ejemplo: **IngresoNumeros**, **MostrarMultiplos**, **CalcularPromedio**. Es decir, cada **subalgoritmo** tiene un propósito único. Si en un programa se repite código, probablemente falta modularizar.
3. El pasaje de **parámetros** permite la comunicación **entre** los subalgoritmos y con el algoritmo, y son de estos dos tipos:

- **Por valor:** se pasa una copia (solo lectura).
- **Por referencia:** se pasa la variable real (lectura y escritura).

Leer la página 4 del manual:

http://servicios.dirinfo.unsl.edu.ar/abm/assets/uploads/materiales/b1e28-2025-pseint_subalgoritmos.pdf

4. El algoritmo ya no “hacerlo todo”. Solo **organiza** el orden de las tareas e **invoca a** los subalgoritmos que resuelven el problema.

Ejemplo práctico:



Enunciado

*Diseñar un algoritmo que permita trabajar con una secuencia de **hasta 10** caracteres ingresados por el usuario. El usuario puede consultar la cantidad de veces que se repite un carácter en particular. Además, el algoritmo debe mostrar la cadena completa de caracteres ingresada.*

Solución SIN modularización aplicada: El programa resuelve el problema dentro de un único bloque de acciones, sin utilizar procedimientos ni funciones.

Versión 1

T1: Declarar las variables necesarias, un arreglo **Cad** de tipo carácter y de dimensión 10, **i** (para recorrer el arreglo), **limS** (para registrar la cantidad de caracteres a ingresar), **cant** (para contar las repeticiones) y **car** (para el carácter a buscar).

T2: Ingresar los datos, Se solicita al usuario cuántos caracteres desea ingresar (hasta 10) y se cargan uno por uno en el arreglo.

T3: Solicitar el carácter a buscar **y Contar** cuantas veces se repite en Cad.

T4: Mostrar el resultado obtenido en t3

T5: Mostrar la cadena completa de caracteres.

```
1  Algoritmo ContarCaracteres_SinModularizacion
2      Definir Cad Como Caracter
3      Definir i, limS, cant Como Entero
4      Definir car Como Caracter
5      Dimension Cad[10]
6
7      // Ingreso de datos
8      Escribir "¿Cuántos caracteres desea ingresar? (máx. 10)"
9      Leer limS
10
11     Para i  $\leftarrow$  1 Hasta limS Con Paso 1 Hacer
12         Escribir "Ingrese el carácter ", i, ":"
13         Leer Cad[i]
14     FinPara
15
16     // Lectura del carácter a buscar
17     Escribir "Ingrese el carácter que desea buscar:"
18     Leer car
19
20     // Cálculo de repeticiones
21     cant  $\leftarrow$  0
22     Para i  $\leftarrow$  1 Hasta limS Con Paso 1 Hacer
23         Si Cad[i] = car Entonces
24             cant  $\leftarrow$  cant + 1
25         FinSi
26     FinPara
27
28     // Mostrar resultados
29     Escribir "El carácter se repite ", cant, " veces."
30     Escribir "La cadena ingresada es: "
31     Para i  $\leftarrow$  1 Hasta limS Con Paso 1 Hacer
32         Escribir Sin Saltar Cad[i]
33     FinPara
34     Escribir "" // salto de línea
35 FinAlgoritmo
36
```

Solución CON modularización aplicada:

La idea central de la modularización es **dividir el problema en partes más simples**, llamadas **subalgoritmos**, de manera que cada uno se encargue de una tarea específica.

Nosotros venimos trabajando con la descomposición en tareas, en la Versión 1, estas tareas pueden implementarse en UN algoritmo y en UNO o más subalgoritmos, donde:

1. **El ALGORITMO pasará a coordinar esas tareas**, en lugar de ejecutarlas directamente. En el algoritmo se definen las variables principales para guardar los datos y se hacen las invocaciones a los subalgoritmos con los parámetros correctos.
2. **Usaremos SUBALGORITMOS** (procedimientos o funciones según corresponda) para cada una de las tareas principales:
 - Un subalgoritmo para el **ingreso de datos** (**IngresoCad**).
 - Un subalgoritmo para el **cálculo de las repeticiones** (**CarRep**).
 - Y otro subalgoritmo **para la muestra de la cadena** (**MostrarCadena**).

Esta organización hace que el código sea **más claro, más fácil de mantener y más reutilizable**. Para que estos subalgoritmos puedan trabajar con la información del programa principal, necesitamos que **se comuniquen mediante parámetros**.

A continuación, vemos cómo QUEDA LA SOLUCIÓN CON MODULARIZACIÓN APLICADA:

Versión 1 del algoritmo principal:

T1: Declarar un arreglo **Cad** de dimensión 10 y de tipo carácter. Declarar la variable **limS** de tipo entero para la cantidad de caracteres que quiere ingresar el usuario. Declarar la variable **cant** de tipo entero.

T2: **Invocar** al procedimiento **IngresoCad con los parámetros** **Cad, limS** (en **limS** se devuelve por referencia la cantidad de caracteres almacenados).

T3: **Invocar** a la función **CarRep con los parámetros** **Cad y limS**. La función retorna la cantidad de repeticiones de un carácter consultado por el usuario en la variable **cant**.

T4: Mostrar la cantidad retornada por la tarea T3.

T5: Invocar un **procedimiento que muestre la cadena de caracteres**.

Analice **dónde** se han definido las variables y cómo se usa la VARIABLE DE RETORNO.

```

1
2 // --- Procedimiento para ingresar los caracteres ---
3 SubAlgoritmo IngresoCad(Cad, limS Por Referencia)
4     Definir i Como Entero
5     //pregunto HASTA cuantos caracteres quiere ingresar
6     Escribir "¿Cuántos caracteres desea ingresar? (máx. 10)"
7     Leer limS
8     //control
9     Mientras limS > 10 o limS < 1 hacer
10         Escribir "error: ¿Cuántos caracteres desea ingresar? (máx. 10)"
11         Leer limS
12     FinMientras
13     //ingreso de datos
14     Para i ← 1 Hasta limS Con Paso 1 Hacer
15         Escribir "Ingrese el carácter ", i, ":"
16         Leer Cad[i]
17     FinPara
18 FinSubAlgoritmo
19
20 // --- Función que cuenta repeticiones ---
21 SubAlgoritmo cant ← CarRep(Cad, limS)
22     Definir car Como Caracter
23     Definir i, cant Como Entero
24     cant ← 0
25     Escribir "Ingrese el carácter que desea buscar:"
26     Leer car
27
28     Para i ← 1 Hasta limS Con Paso 1 Hacer
29         Si Cad[i] = car Entonces
30             cant ← cant + 1
31         FinSi
32     FinPara
33 FinSubAlgoritmo
34
35 // --- Procedimiento para mostrar la cadena completa ---
36 SubAlgoritmo MostrarCadena(Cad, limS)
37     Definir i Como Entero
38     Escribir "La cadena ingresada es: "
39     Para i ← 1 Hasta limS Con Paso 1 Hacer
40         Escribir Cad[i]
41     FinPara
42 FinSubAlgoritmo
43
44 Algoritmo ContarCaracteres
45     Definir Cad Como Caracter
46     Definir limS, cant Como Entero
47     Dimension Cad[10]
48
49     // T2: Ingreso de datos
50     IngresoCad(Cad, limS)
51
52     // T3: Cálculo de repeticiones
53     cant ← CarRep(Cad, limS)
54
55     // T4: Mostrar resultado
56     Escribir "El carácter se repite ", cant, " veces."
57
58     // T5: Mostrar la cadena completa
59     MostrarCadena(Cad, limS)
60 FinAlgoritmo
61

```