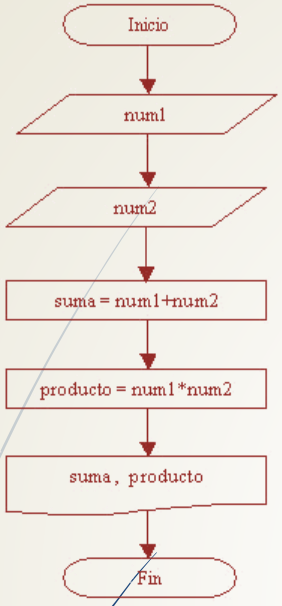
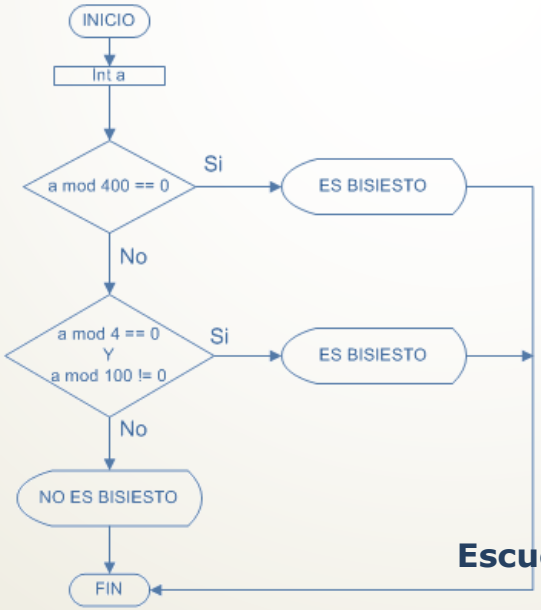
Técnico Informático



Programación Estructurada

Programación Modular



Escuela de Minas "Dr. Horacio Carrillo"
Universidad Nacional de Jujuy

Índice

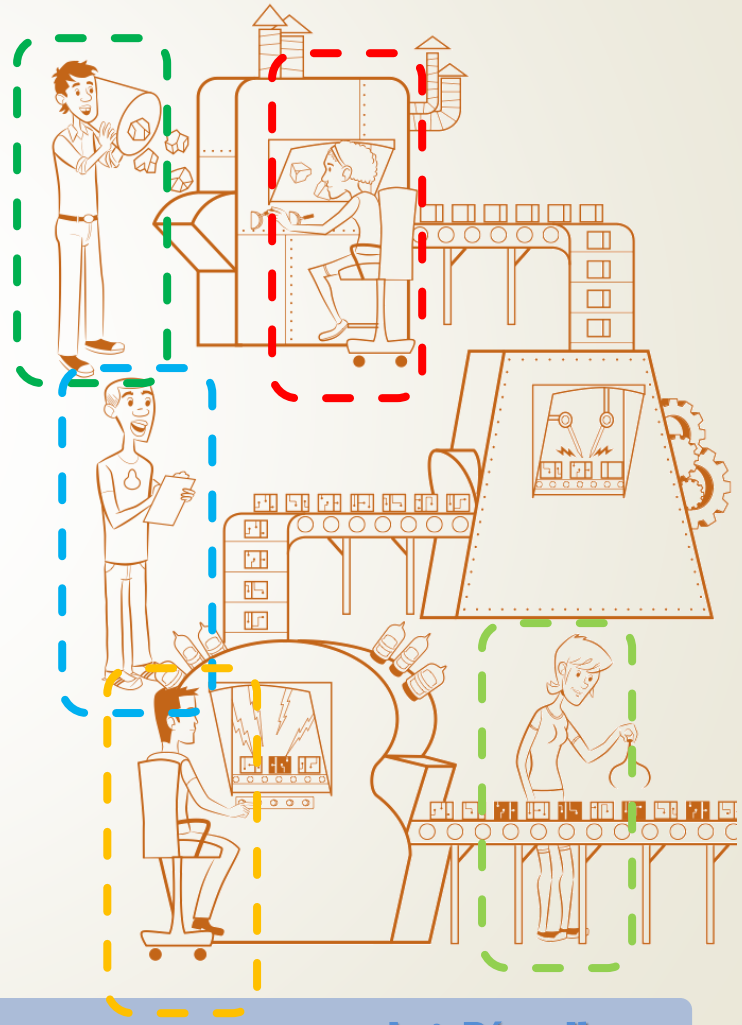
- Programación Modular
- Funciones
- Procedimientos
- Comunicación entre módulos
  - Pasaje de Parámetros
- Variables Locales y Globales
- Ocultamiento y Protección

Estructura de Programa

- Cabecera de Programa
- Declaración de Constantes
- Declaración de Tipos
  - Tipos definidos por el programador
- Declaración de Variables
- **Declaración de Procedimientos y Funciones**
- Programa Principal

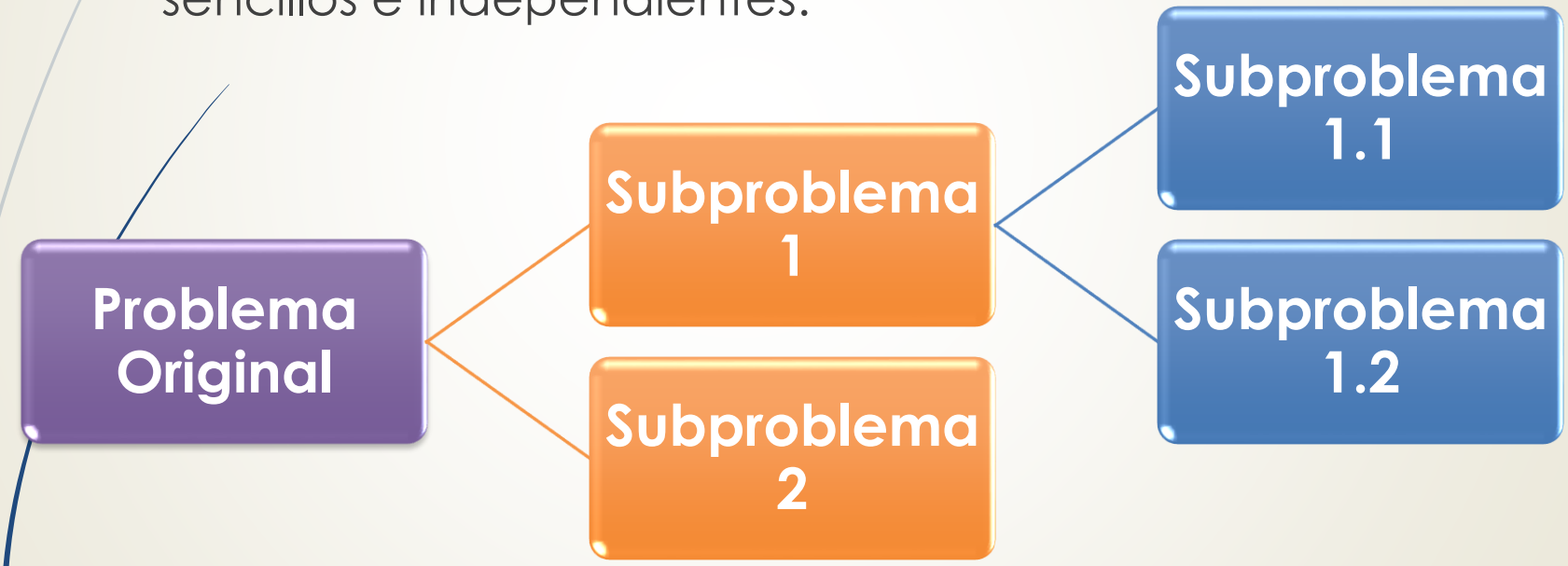
Complejidad de un problema

- Los problemas sencillos pueden resolverse usando unas pocas instrucciones simples.
- Los problemas difíciles se resuelven combinando instrucciones de forma compleja.
- Para reducir la complejidad de un programa, éste puede dividirse en varias unidades de trabajo.



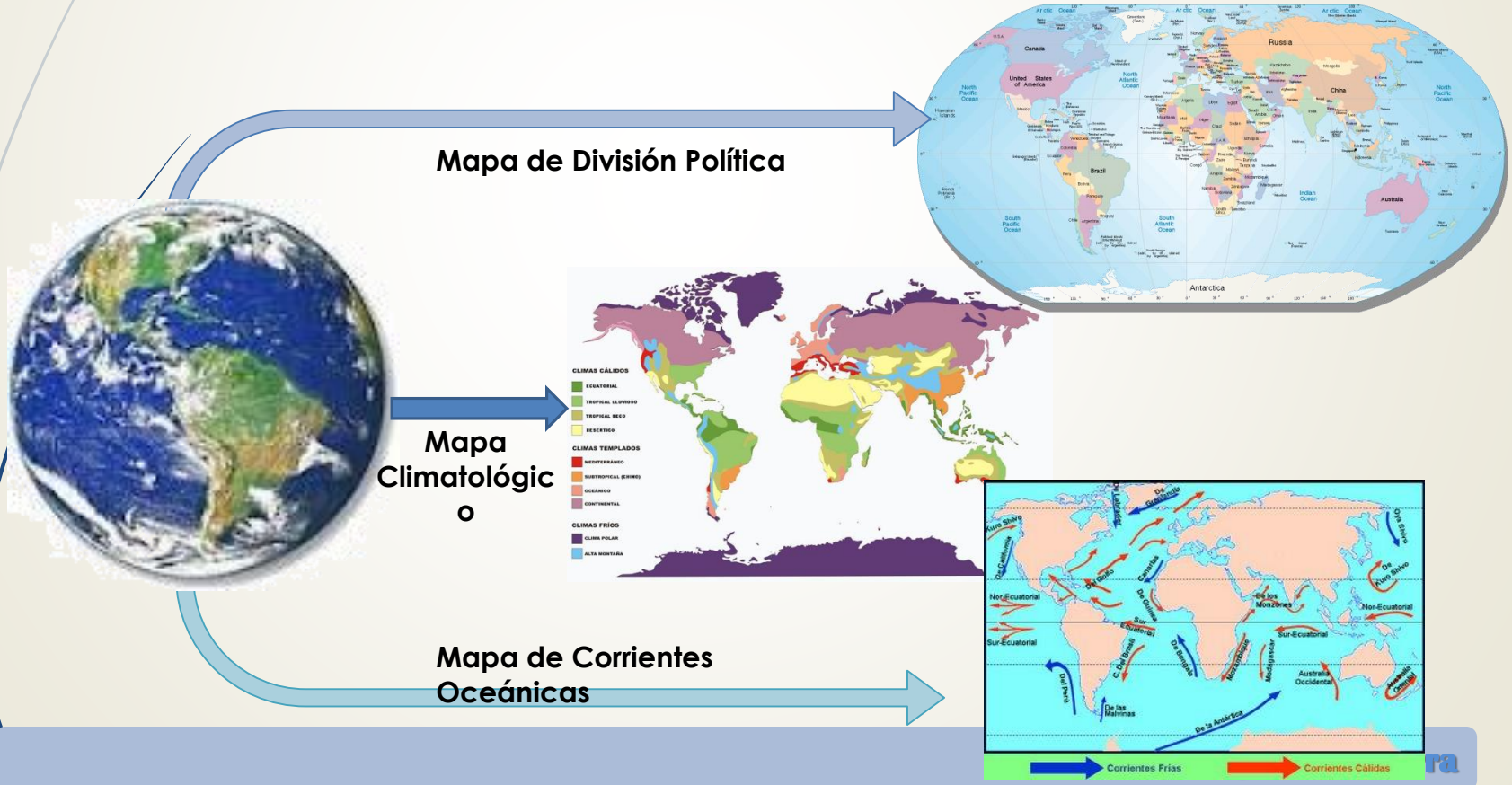
Programación Modular (1)

- Descomposición de problemas
  - Un problema complejo puede dividirse en problemas sencillos e independientes.



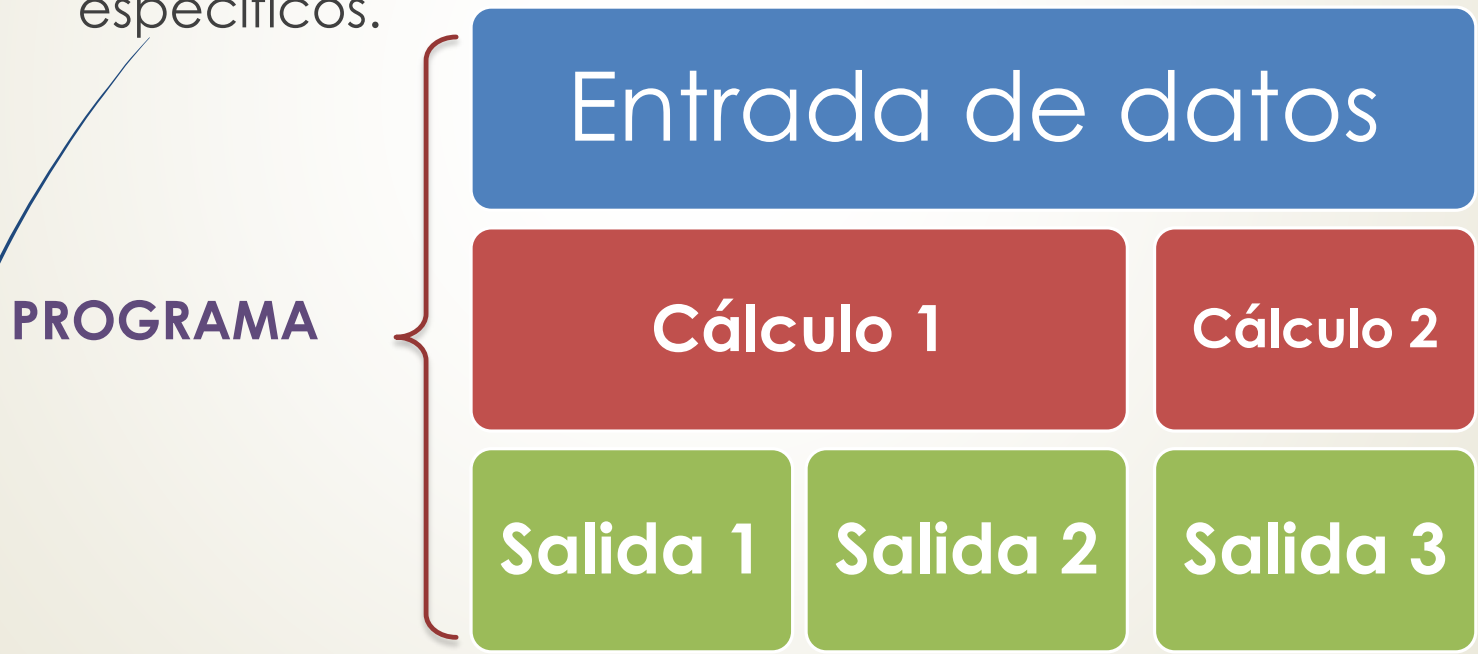
Programación Modular (2)

- Abstracción
  - Permite representar los objetos relevantes del problema.



Programación Modular (3)

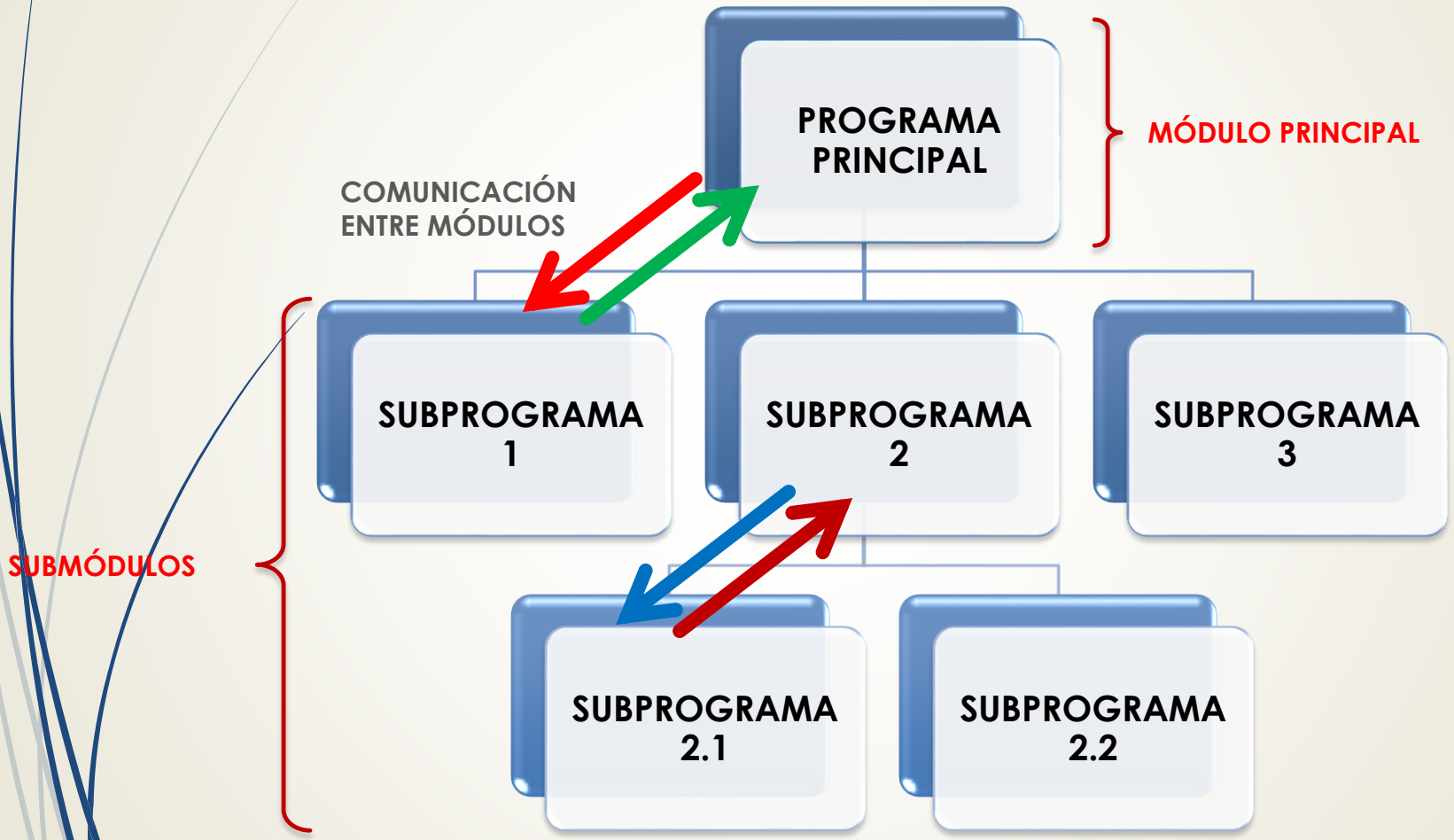
- Módulos
  - Un programa puede estar formado por partes independientes que resuelven subproblemas específicos.



Programación Modular (4)

- La programación modular
  - es un método de diseño flexible
  - permite dividir un programa en subprogramas
- Subprogramas o módulos
  - pueden analizarse, codificarse y probarse por separado
  - el módulo principal controla el flujo de acciones
  - se clasifican en Procedimientos y Funciones

Programas Modulares (1)

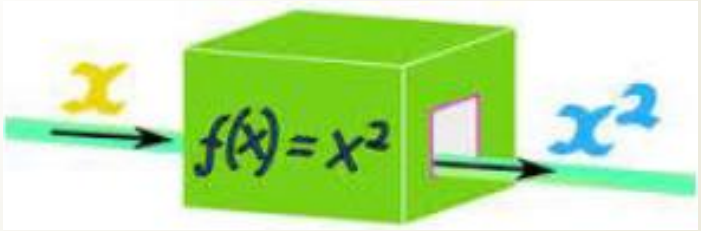


Programas Modulares (2)

- Entradas: se conoce el conjunto de datos con los que trabajará el módulo.
- Propósito: se conoce el objetivo del módulo (qué hace).
- Salidas: se conoce el resultado que generará el módulo.

Funciones

- Una función es un módulo o subprograma que toma una lista de valores llamados argumentos o parámetros y devuelve un único valor.
- Las funciones se definen de un tipo de dato (entero, real, carácter, lógico).
- Las funciones pueden ser internas o definidas por el usuario.



Declaración de funciones

```
FUNCIÓN Nombre_función (Parámetros formales): Tipo_de_Función
VARIABLES
    Variables_de_la_Función
INICIO
    ACCIONES
    Nombre_función←resultado_de_la_función
FIN
```

- ✓ *Nombre_función:* especifica el nombre de la función.
- ✓ *Parámetros Formales:* son los valores que recibe la función y que se usarán en el cálculo.
- ✓ *Tipo_de_Función:* la función puede ser entera, real, carácter, lógica.
- ✓ *Variables_de_la_Función:* son las variables de la función, están definidas para la función y desaparecen cuando ésta finaliza su ejecución.
- ✓ *ACCIONES:* sentencias secuenciales, selectivas o repetitivas que implementan la operación.
- ✓ *Nombre_función ←resultado_de_la_función:* el resultado del cálculo de la función se asigna al nombre de la función y se retorna al programa que la invocó.

Invocación de funciones

- Un función puede invocarse:

```
Variable←nombre_función(parámetros_actuales)
```

```
ESCRIBIR "Resultado:", nombre_función(parámetros_actuales)
```

- Al invocar una función:
  1. A cada parámetro formal se le asigna el valor de su correspondiente parámetro actual.
  2. Se ejecuta el cuerpo de acciones de la función.
  3. Se asigna el resultado a la función y se retorna al punto de llamada.

Procedimientos

- Un procedimiento o subrutina es un subprograma que ejecuta un proceso específico.
- Los procedimientos, a diferencia de las funciones, no tienen asociado un valor.
- Los procedimientos pueden recibir parámetros para llevar acabo su trabajo, e incluso modificar éstos si es necesario.

Declaración de Procedimientos

```
PROCEDIMIENTO Nombre_procedimiento (Parámetros formales)
VARIABLES
   Variables_del_Procedimiento
INICIO
   ACCIONES
FIN
```

- ✓*Nombre_procedimiento:* especifica el nombre del procedimiento.
- ✓*Parámetros Formales:* especifica los valores que recibe el procedimiento, con los que realizará algún procesamiento.
- ✓*Variables_del_Procedimiento:* especifica las variables del procedimiento, éstas sólo están definidas para el procedimiento y desaparecen cuando finaliza su ejecución.
- ✓*ACCIONES:* sentencias secuenciales, selectivas o repetitivas que realizan la operación definida para el procedimiento.

Invocación de Procedimientos

- Un procedimiento puede invocarse:

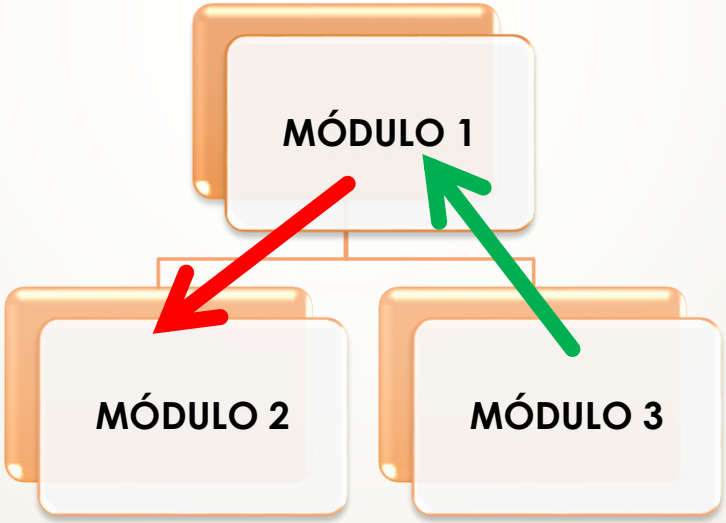
```
Nombre_Procedimiento(Parámetros_actuales)
```

- Al invocar un procedimiento:

1. Los parámetros actuales sustituyen a los parámetros formales.
2. El cuerpo de la declaración del procedimiento sustituye el llamado del procedimiento.
3. Se ejecutan las acciones escritas por el código resultante.

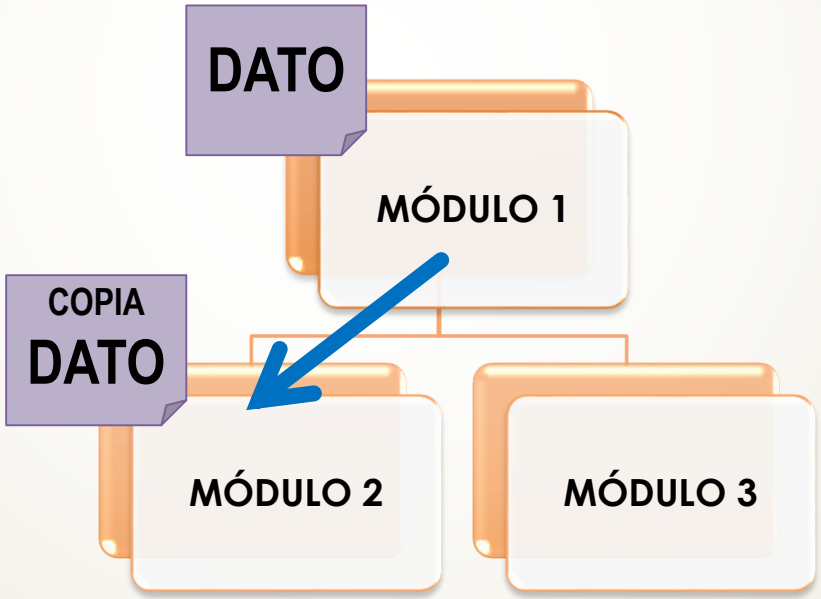
Comunicación entre módulos

- Los datos que usan los módulos de un programa se comunican a éstos cuando son invocados. Esta comunicación se denomina *Pasaje de Parámetros*.



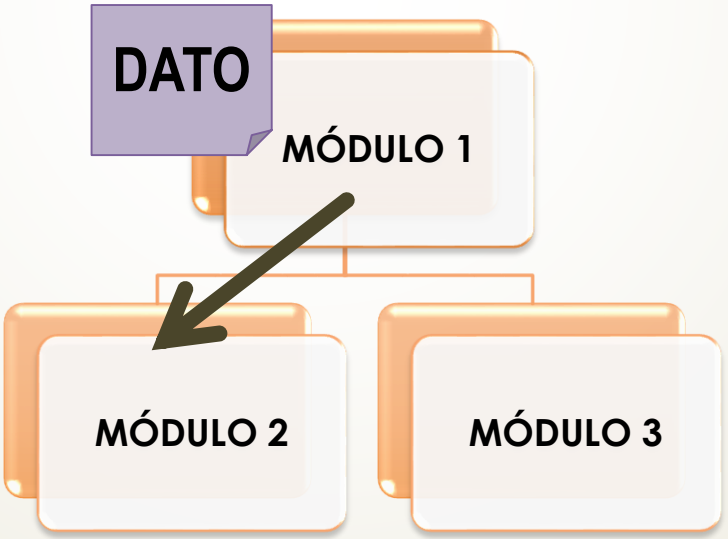
Pasaje de Parámetros (1)

- Por Valor: el módulo trabaja con copias de los datos originales. Estos parámetros se conocen como de entrada (E).



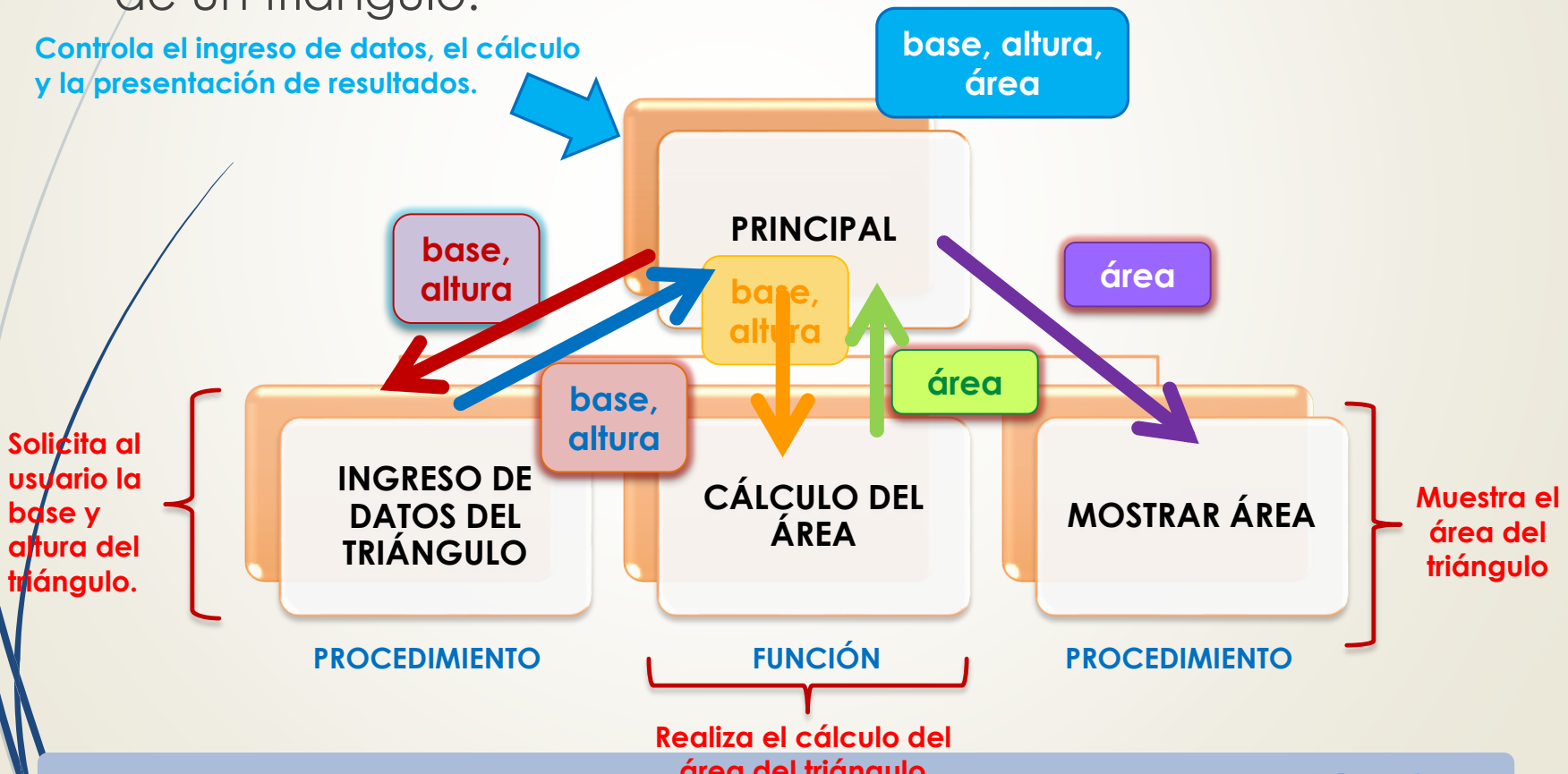
Pasaje de Parámetros (2)

- Por Referencia: el módulo trabaja con los datos originales y cualquier modificación altera los datos del programa que invocó al módulo. Estos parámetros se conocen como de entrada y salida (E/S).



Pasaje de Parámetros. Ej. (1)

- Ejemplo: Diseñe un programa modular que calcule el área de un triángulo.



Pasaje de Parámetros. Ej. (2)

- Ejemplo: Diseñe un programa modular que calcule el área de un triángulo.

```
PROGRAMA calculo_triangulo
VARIABLES
    base, altura, area: real
                    …
INICIO
  Leer_datos(base,altura)
  area<-Calculo_area(base,altura)
  Mostrar_area(area)
FIN
```

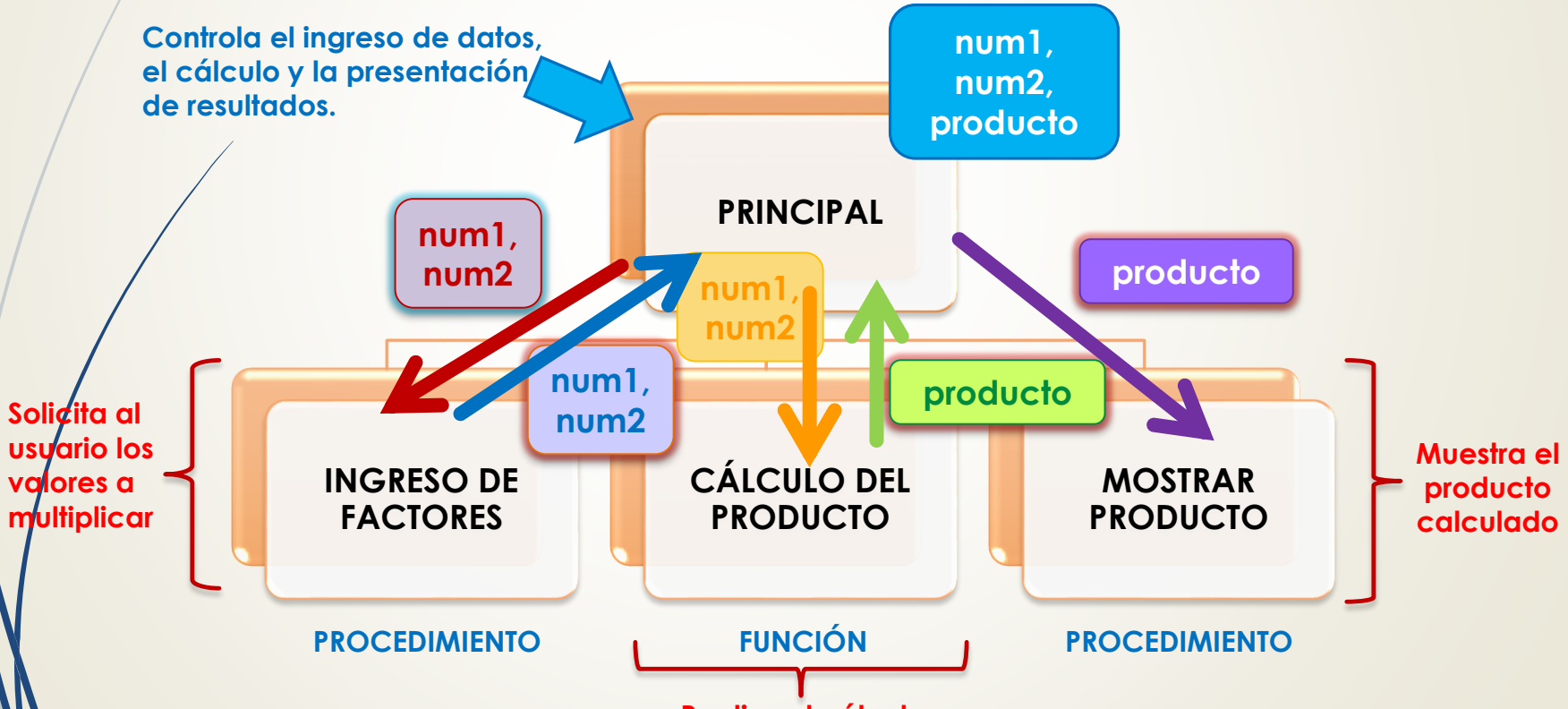
```
PROCEDIMIENTO Leer_datos(E/S b: real, E/S h: real)
INICIO
  ESCRIBIR 'Ingrese la base del triángulo:'
  LEER b
  ESCRIBIR 'Ingrese la altura del triángulo:'
  LEER h
FIN
```

```
FUNCIÓN Calculo_area(E b:real, E h:real): real
INICIO
   Calculo_area<-b * h / 2
FIN
```

```
Procedimiento mostrar_area(E area:real)
INICIO
   ESCRIBIR 'El área calculada es:', area
FIN
```

Pasaje de Parámetros. Ej. (3)

- Ejemplo: Diseñe un programa modular que calcule el producto de 2 números mediante sumas sucesivas.



Pasaje de Parámetros. Ej. (4)

- Ejemplo: Diseñe un programa modular que calcule el producto de 2 números mediante sumas sucesivas.

```
PROGRAMA calculo_producto
VARIABLES
   factor1, factor2, producto: ENTERO
                        …
INICIO
  leer_datos(factor1,factor2)
  producto<-producto_sumas(factor1,factor2)
  mostrar_producto(producto)
FIN
```

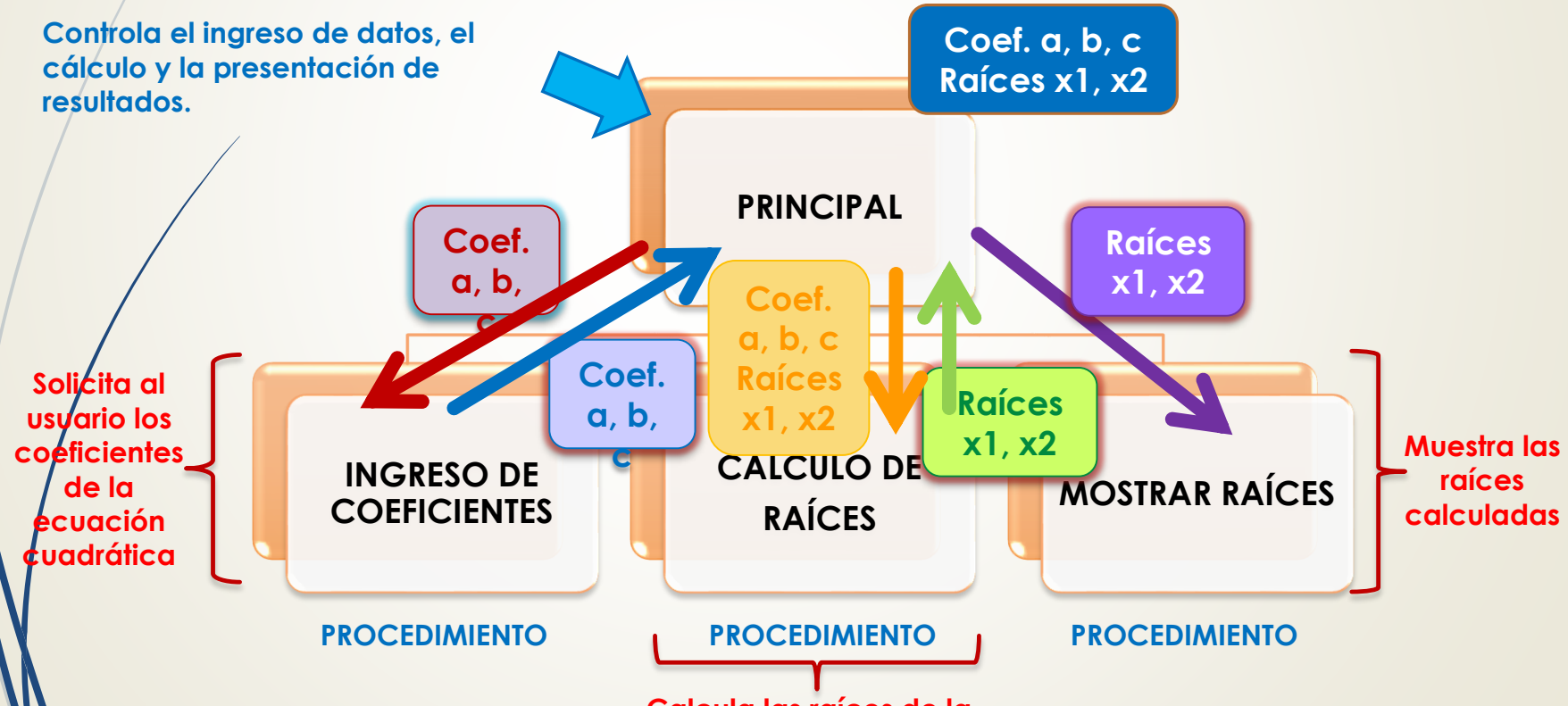
```
PROCEDIMIENTO leer_datos(E/S a:ENTERO, E/S b: ENTERO)
INICIO
  ESCRIBIR "Ingrese primer factor: "
  LEER a
  ESCRIBIR "Ingrese segundo factor: "
  LEER b
FIN
```

```
FUNCIÓN producto_sumas(E a:ENTERO, E b:ENTERO):ENTERO
VARIABLES
  i,p: ENTERO
INICIO
    p<-0;
    PARA i DESDE 1 HASTA b CON PASO 1 HACER
       p<-p+a
    FIN_PARA
    producto_sumas<-p
FIN
```

```
PROCEDIMIENTO mostrar_producto(E prod:ENTERO)
INICIO
  ESCRIBIR “Producto: “, prod
FIN
```

Pasaje de Parámetros. Ej. (5)

- Ejemplo: Diseñe un programa modular que calcule las raíces de una ecuación cuadrática.



Pasaje de Parámetros. Ej. (6)

- Ejemplo: Diseñe un programa modular que calcule las raíces de una ecuación cuadrática.

```
PROGRAMA calculo_raices
VARIABLES
      ca, cb, cc, x1, x2: REAL
INICIO
   leer_datos(ca,cb,cc)
   raices(ca,cb,cc,x1,x2)
   mostrar_raices(x1,x2)
FIN
```

```
PROCEDIMIENTO leer_datos (E/S a: REAL, E/S b: REAL, E/S c:REAL)
INICIO
  ESCRIBIR "Ingrese coef. cuadratico: “
  LEER a
  ESCRIBIR "Ingrese coef. lineal: "
  LEER b
  ESCRIBIR "Ingrese coef. indep.: "
  LEER c
FIN
```

```
PROCEDIMIENTO raices(E a:REAL,E b:REAL, E c:REAL,E/S r1:REAL, E/S r2: REAL)
INICIO
      r1<-(-b + (b^2-4*a*c)^(1/2))/(2*a)
      (2*a)
```

```
PROCEDIMIENTO mostrar_raices ( E r1: REAL, E r2:REAL)
INICIO
   ESCRIBIR "Raiz 1: “, r1
   ESCRIBIR "Raiz 2: “, r2
FIN
```

Variables Locales

- Una variable local es aquella que está declarada y definida dentro de un subprograma.
- El significado de las variables locales se confina al subprograma que las contiene.
- Una variable local no puede ser accedida por otros módulos del programa.

Variables Globales

- Una variable global es aquella que está declarada y definida en el programa principal.
- Las variables globales son conocidas por todos los módulos del programa.
- Una variable global puede ser accedida por cualquier módulo del programa.

¡Cuidado! La manipulación de variables globales en los módulos puede ocasionar modificaciones accidentales y generar errores en el programa.

Ocultamiento y Protección

- *Data Hidding* significa que los datos relevantes para un módulo deben ocultarse a otros módulos.
- Esto evita que en el programa principal se declaren datos que sólo son relevantes para algún módulo en particular y, además, se protege la integridad de los datos.

Bibliografía

- Sznajdleder, Pablo Augusto. Algoritmos a fondo. Alfaomega. 2012.
- López Román, Leobardo. Programación estructurada y orientada a objetos. Alfaomega. 2011.
- De Giusti, Armando *et al*. Algoritmos, datos y programas, conceptos básicos. Editorial Exacta, 1998.
- Joyanes Aguilar, Luis. Fundamentos de Programación. Mc Graw Hill. 1996.
- Joyanes Aguilar, Luis. Programación en Turbo Pascal. Mc Graw Hill. 1990.