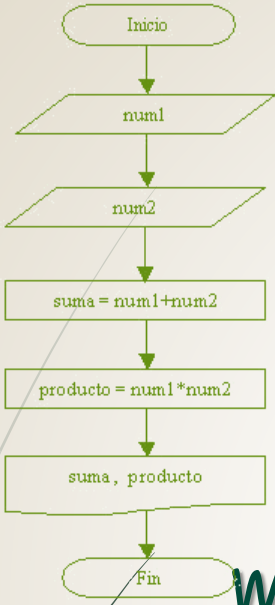
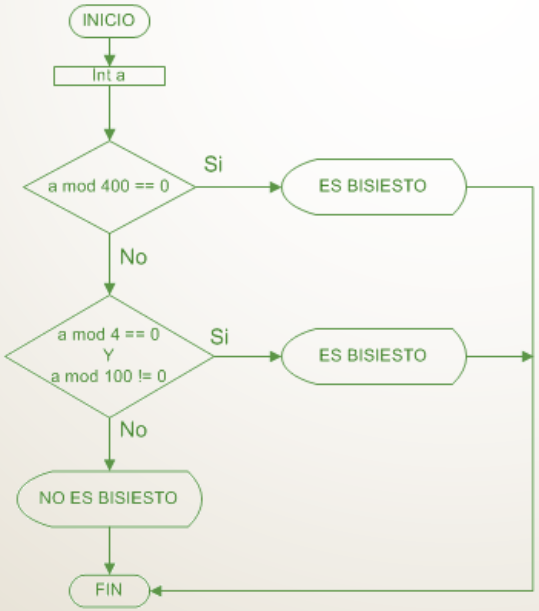


Programación Estructurada

MODULARIDAD: PROCEDIMIENTOS



Facultad de Ingeniería
Universidad Nacional de Jujuy

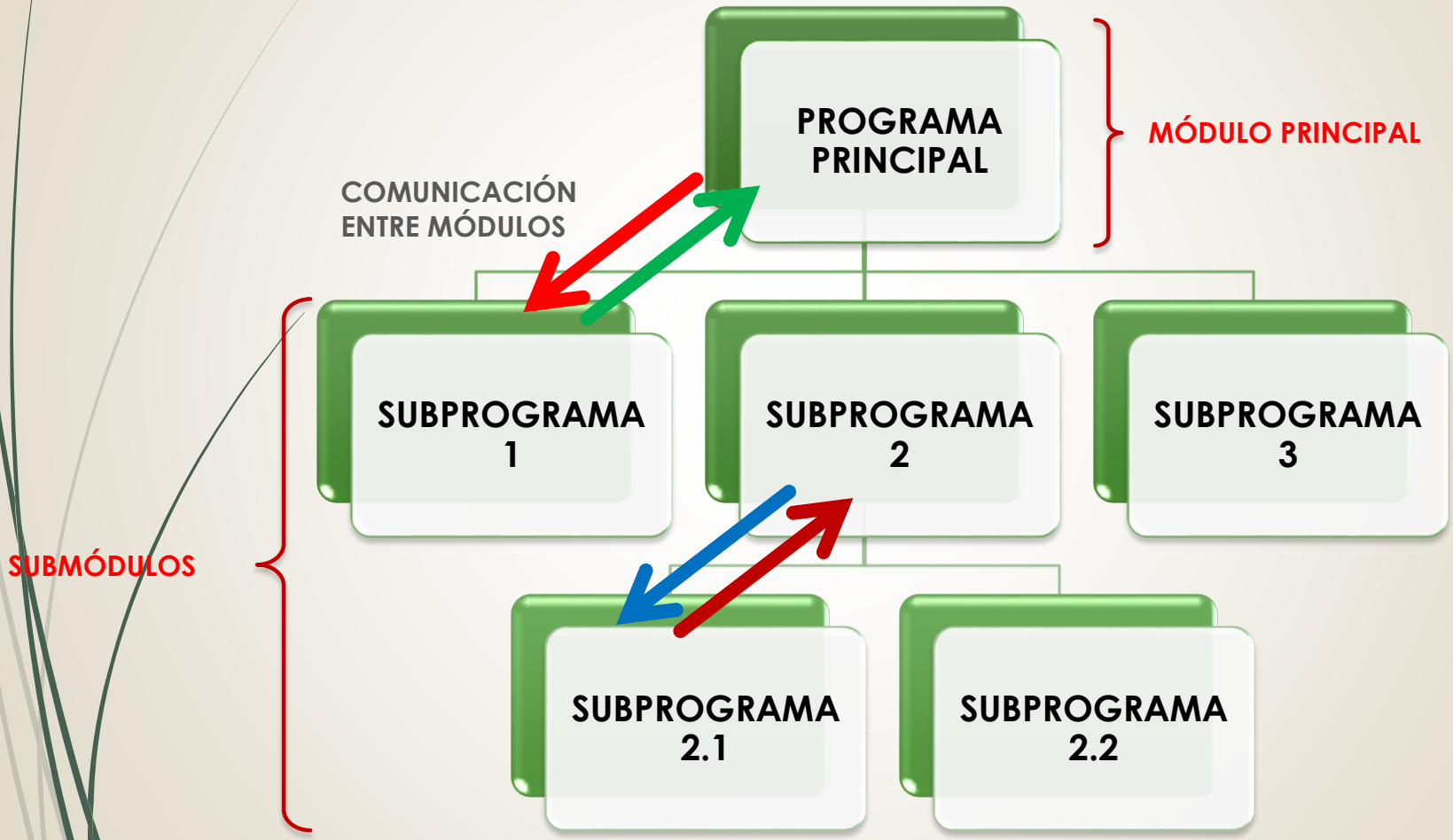
Complejidad de Problemas

- Problemas Complejos
  - Requieren de múltiples tareas o acciones para su resolución
  - Cada tarea o acción puede ser a su vez compleja
- ¿Cómo reducir la complejidad?
  - El problema puede dividirse en subproblemas
  - Las tareas de resolución pueden dividirse en subtareas

Programación Modular

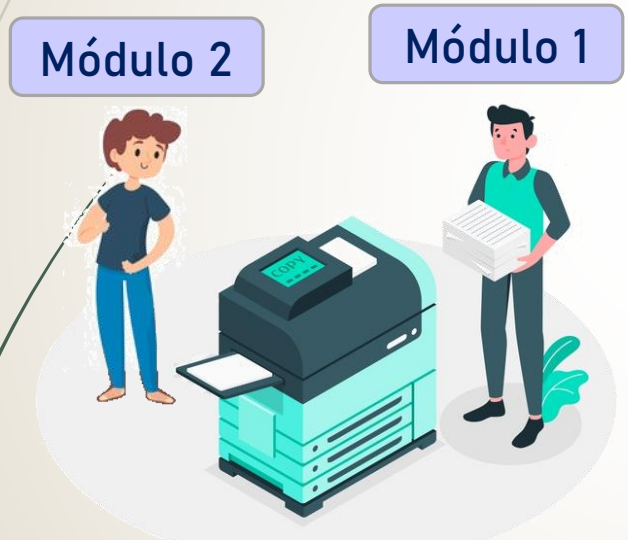
- La programación modular es un método de diseño
  - tratamiento de problemas complejos
  - se basa en descomposición de problemas, abstracción y módulos
  - división un programa en subprogramas (división de tareas)
- Subprogramas o módulos
  - pueden analizarse, codificarse y probarse por separado
  - el módulo principal controla el flujo de acciones
  - se clasifican en **Procedimientos** y **Funciones**

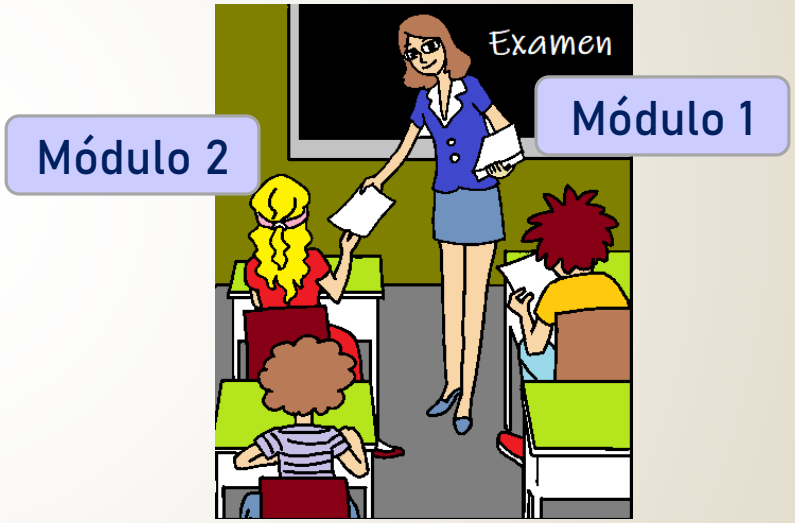
Programas Modulares



Comunicación entre módulos

- La comunicación entre módulos se realiza mediante *Pasaje de Parámetros*: por valor y por referencia.





Función factorial (E n: ENTERO): ENTERO

int factorial (int n)

Procedimiento menu (E/S op: CARACTER)

void menu (char &opcion)

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 (1)

```
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.

Declaración de Procedimientos (2)

```
void nombre_procedimiento (parámetros formales)
{
   tipo_dato nombre_variables; //variables del proced.
   ACCIONES;
}
```

- **void**: los procedimientos se definen de tipo void.
- **parámetros formales**: indica los valores (y sus tipos) que utilizará el procedimiento.
- **variables del procedimiento**: son las variables creadas sólo para el procedimiento (locales).
- **ACCIONES**: instrucciones secuenciales, selectivas y repetitivas.

Invocación de Procedimientos

- Un procedimiento puede invocarse:

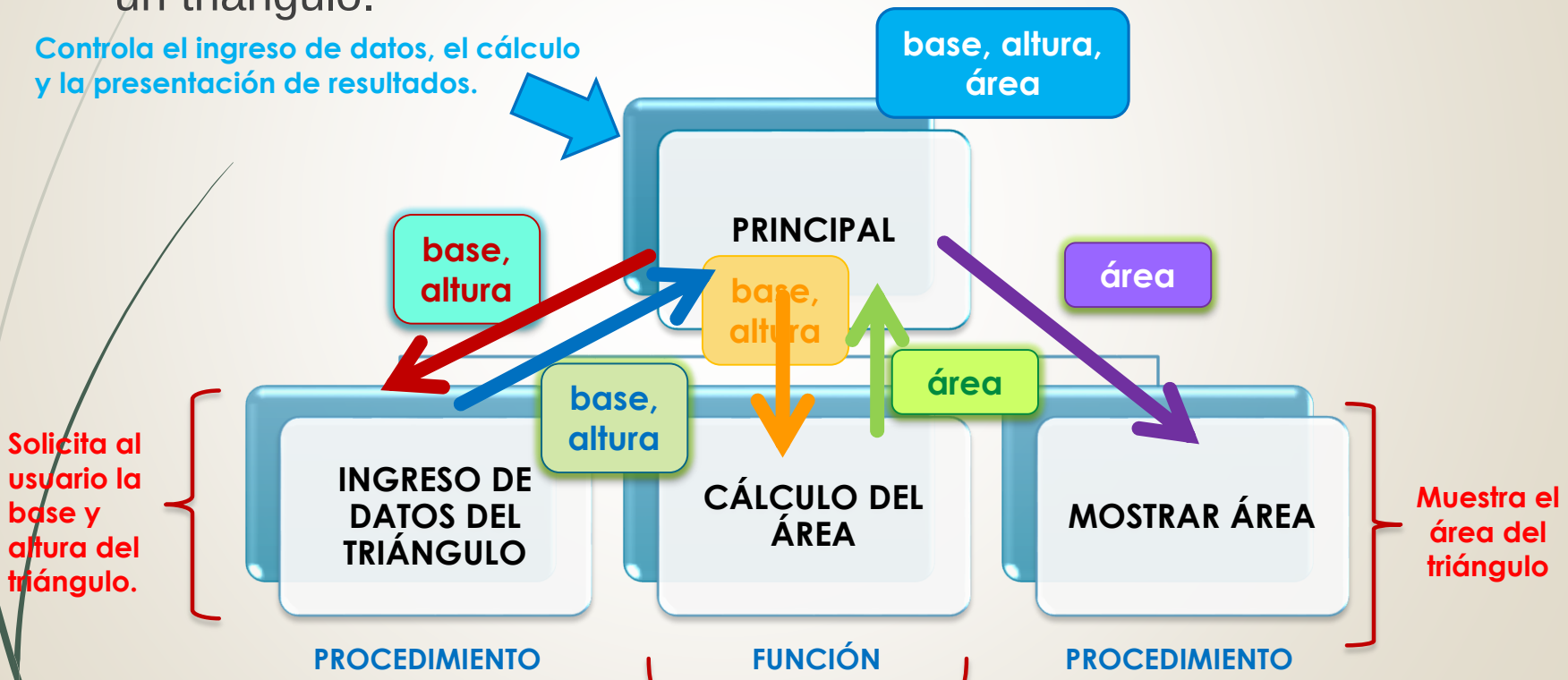
```
Nombre_Procedimiento(Parámetros_actuales)
```

```
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.

Ejemplo (1)

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



Ejemplo (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
```

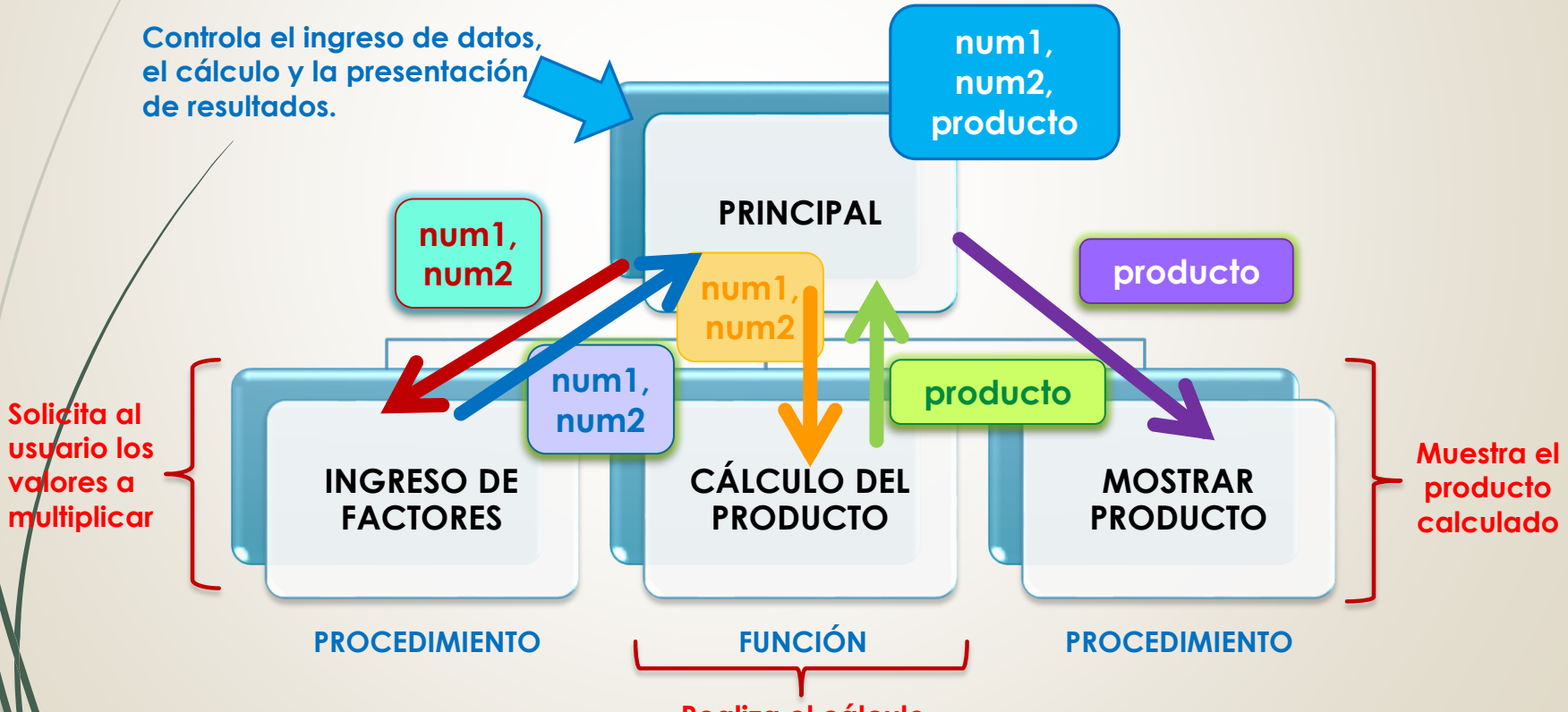
```
PROCEDIMIENTO Leer_datos(E/S b: real, E/S h: real)
INICIO
  ESCRIBIR 'Ingrese la base del triángulo:'
  LEER b
  ESCRIBIR 'Ingrese la altura del triángulo:'
  LEER h
FIN
```

```
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
```

Ejemplo (3)

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



Ejemplo (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
```

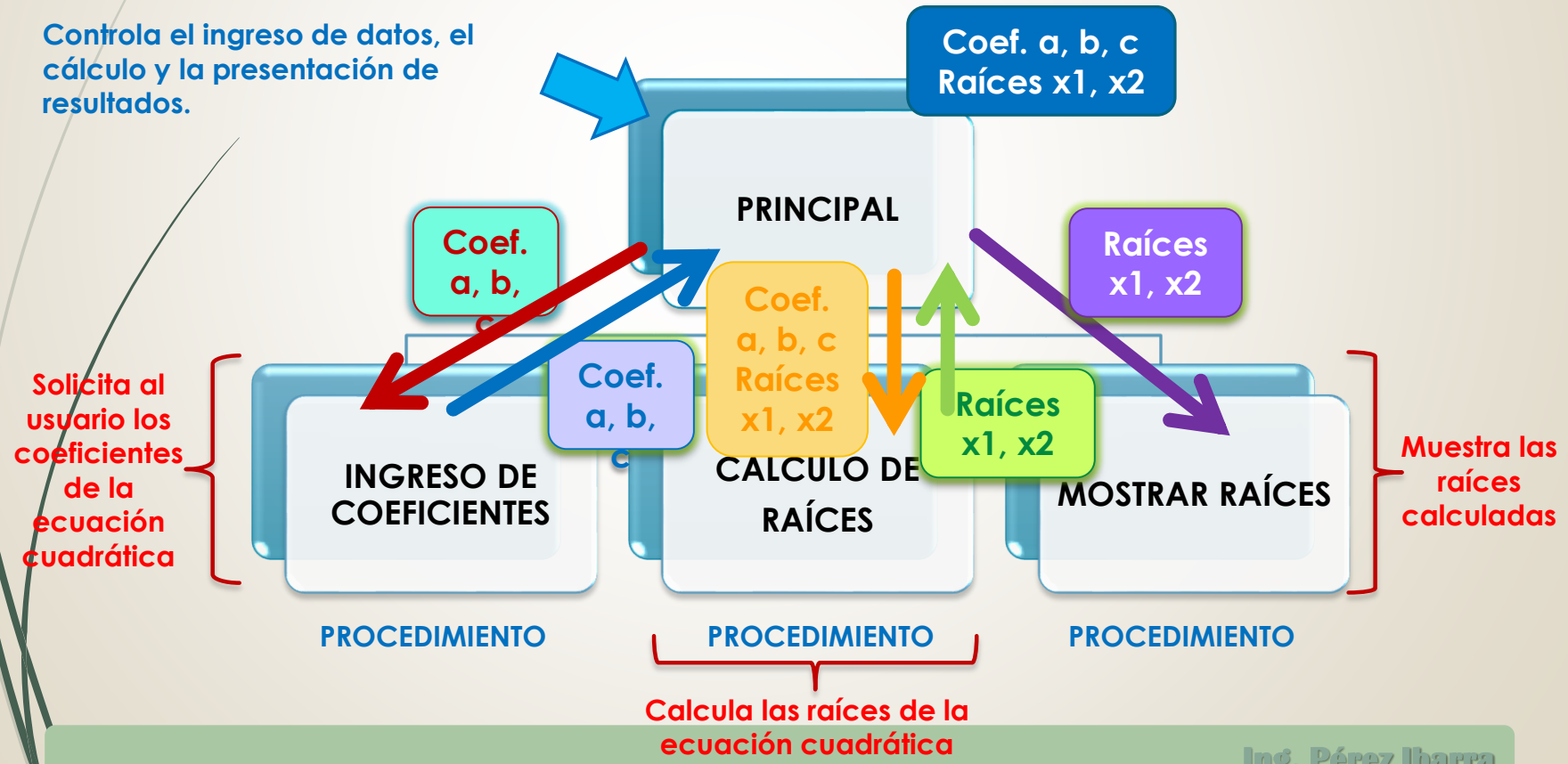
```
PROCEDIMIENTO leer_datos(E/S a:ENTERO, E/S b: ENTERO)
INICIO
  ESCRIBIR "Ingrese primer factor: "
  LEER a
  ESCRIBIR "Ingrese segundo factor: "
  LEER b
FIN
```

```
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
```

Ejemplo (5)

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



Ejemplo (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.