

- 1 Concepto de Modularización
- 2 Procedimientos y Funciones
- 3 Comunicación entre módulos
- 4 Ejercitación

1 Concepto de Modularización

Etapas de resolución de un problema por computadora

Problema
Solución

*Problema del
Mundo Real*

Análisis

Complejos
Extensos
Modificables

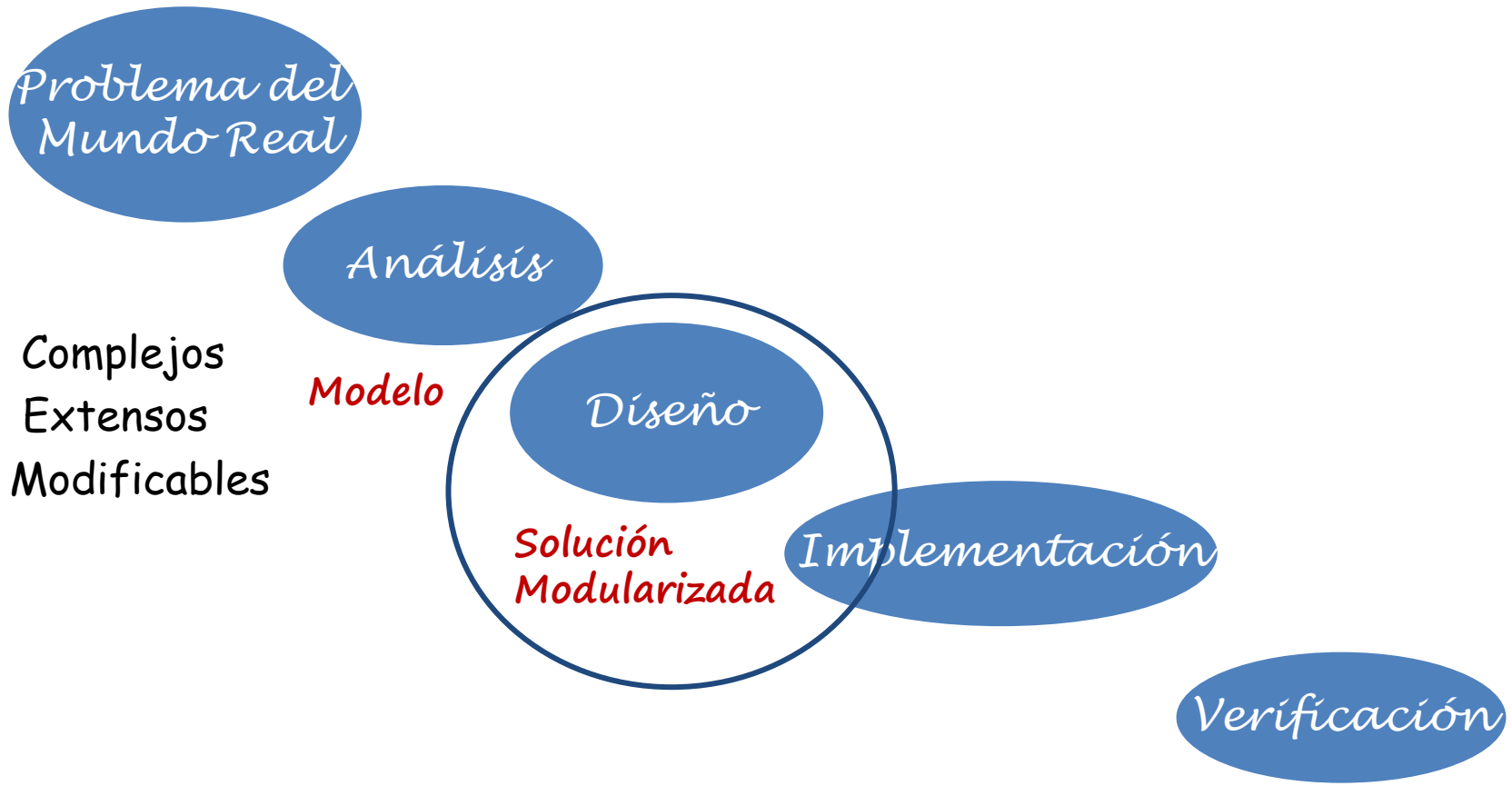
Modelo

Diseño

*Solución
Modularizada*

Implementación

Verificación



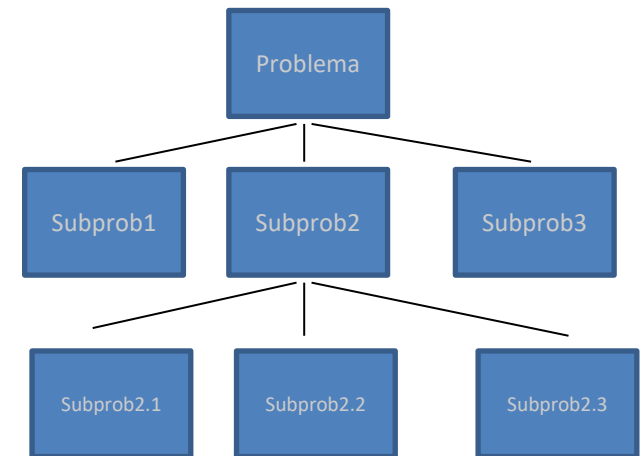
Metodología de diseño Top Down

Principio de "Divide y vencerás"

Descomponer el problema en partes (subproblemas) mas simples

Al descomponer un problema se debe tener en cuenta:

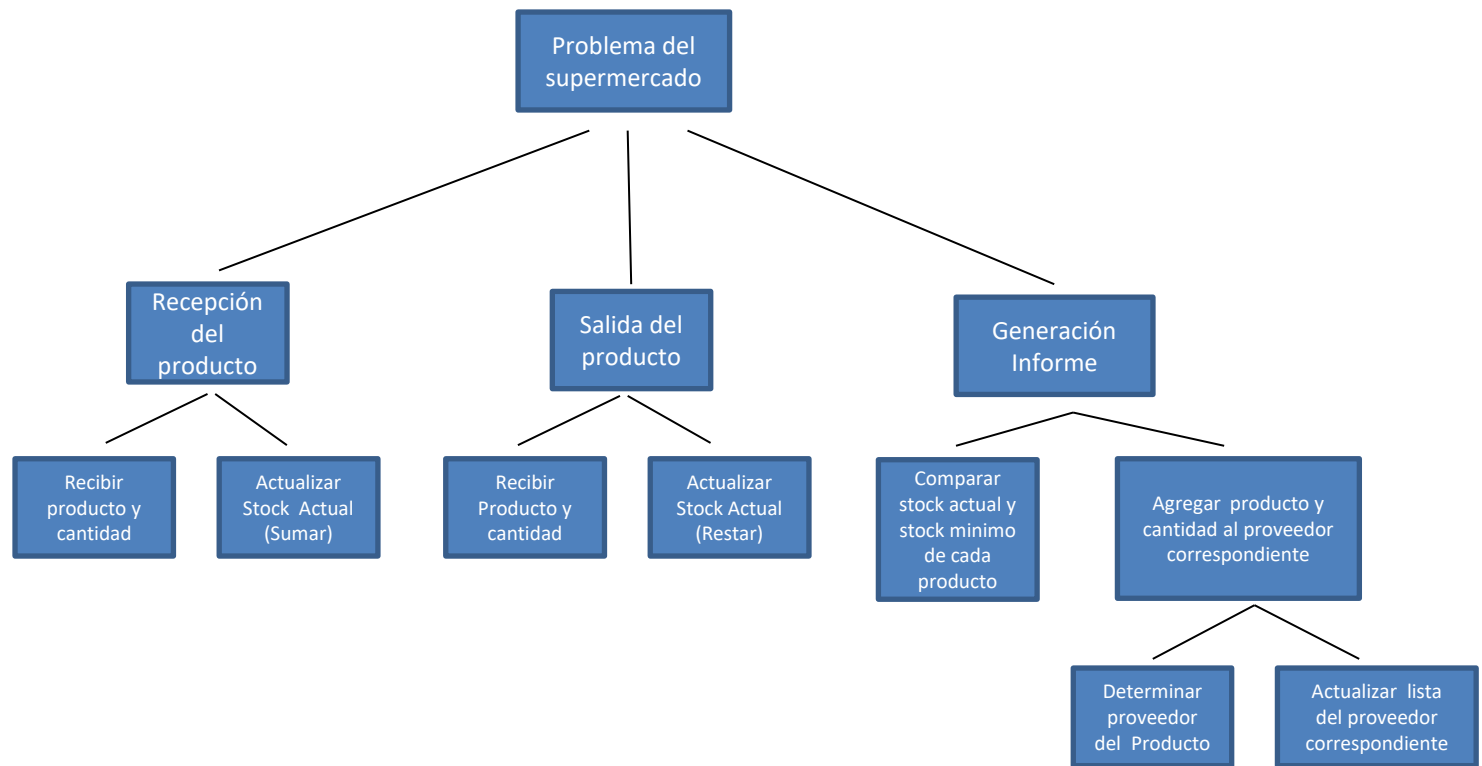
- Que cada subproblema resuelva una parte "bien" simple.
- Que cada subproblema pueda resolverse independientemente
- Que las soluciones a los subproblemas deben combinarse para resolver el problema original

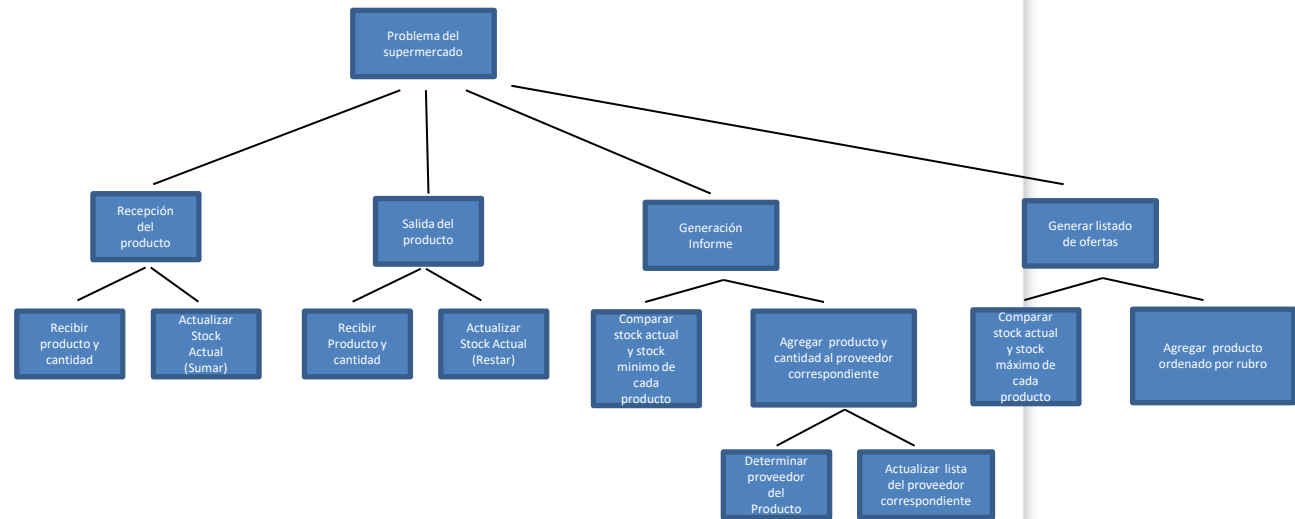


¿Cuándo se detiene la descomposición del problema?

Metodología de diseño Top Down

Si retomamos el problema planteado en la clase anterior en el cual el dueño de un supermercado necesita disponer de un informe con los productos que se deben reponer ordenado por proveedor, una posible descomposición bien simple podría ser la siguiente:





Permite distribuir el trabajo

*Ventajas de la
descomposición
del problema*

Favorece el mantenimiento
correctivo

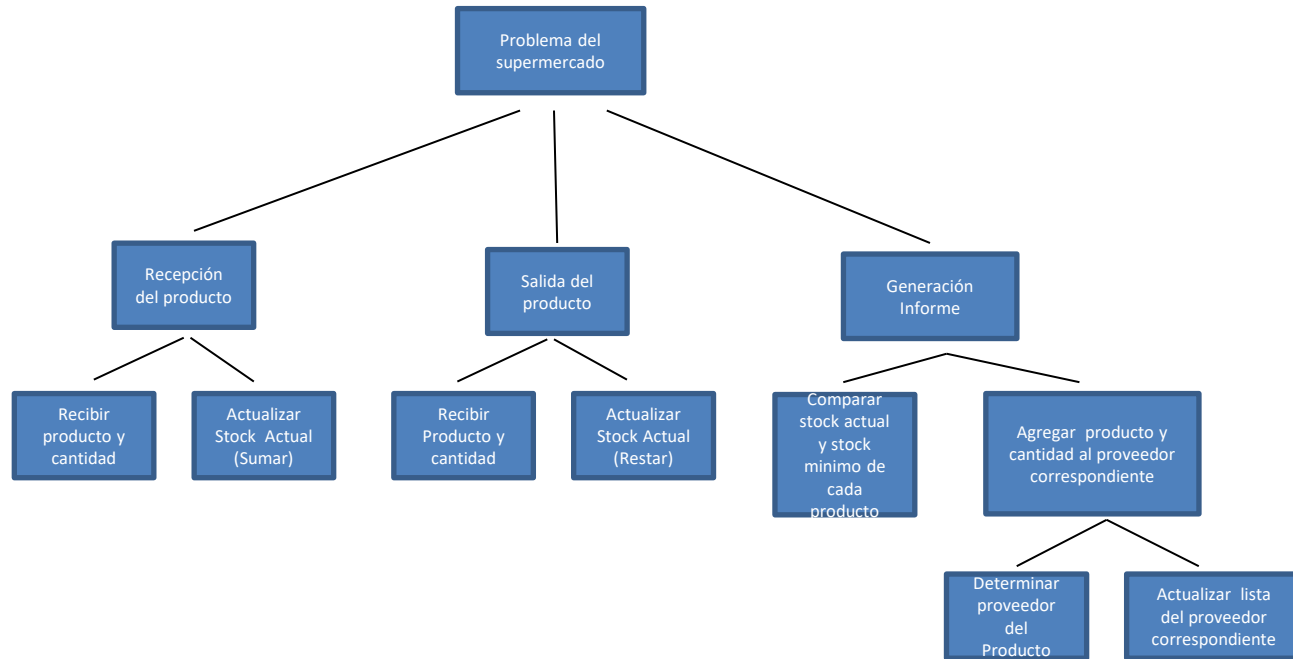
Facilita la reutilización dentro del
mismo problema o en otro similar

Aumenta la legibilidad

Facilita el crecimiento de los
sistemas

Retomando el concepto de Modularización

La tarea de **Modularizar** implica dividir un problema en partes. Se busca que cada parte realice una tarea simple y pueda resolverse de manera independiente a las otras tareas.



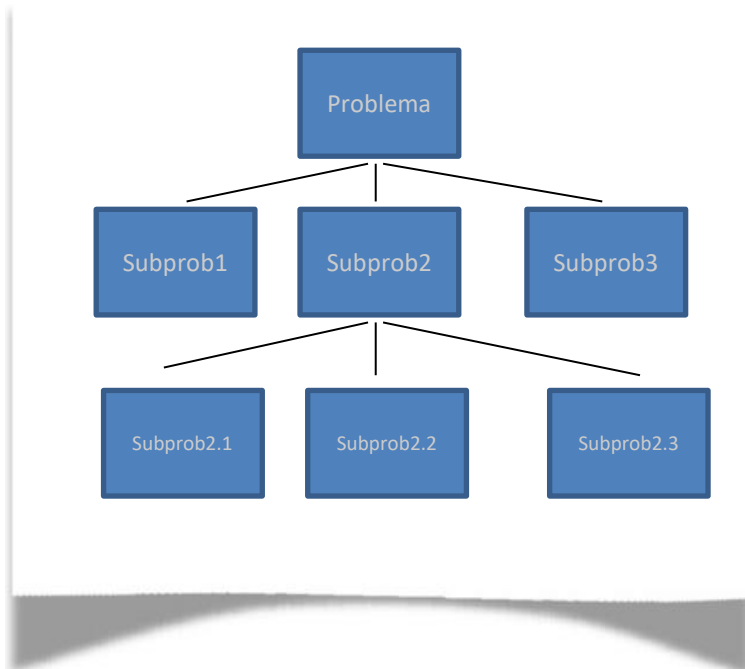
Cada uno de los módulos

Resuelve un subproblema particular

Define el conjunto de acciones

Define los datos necesarios

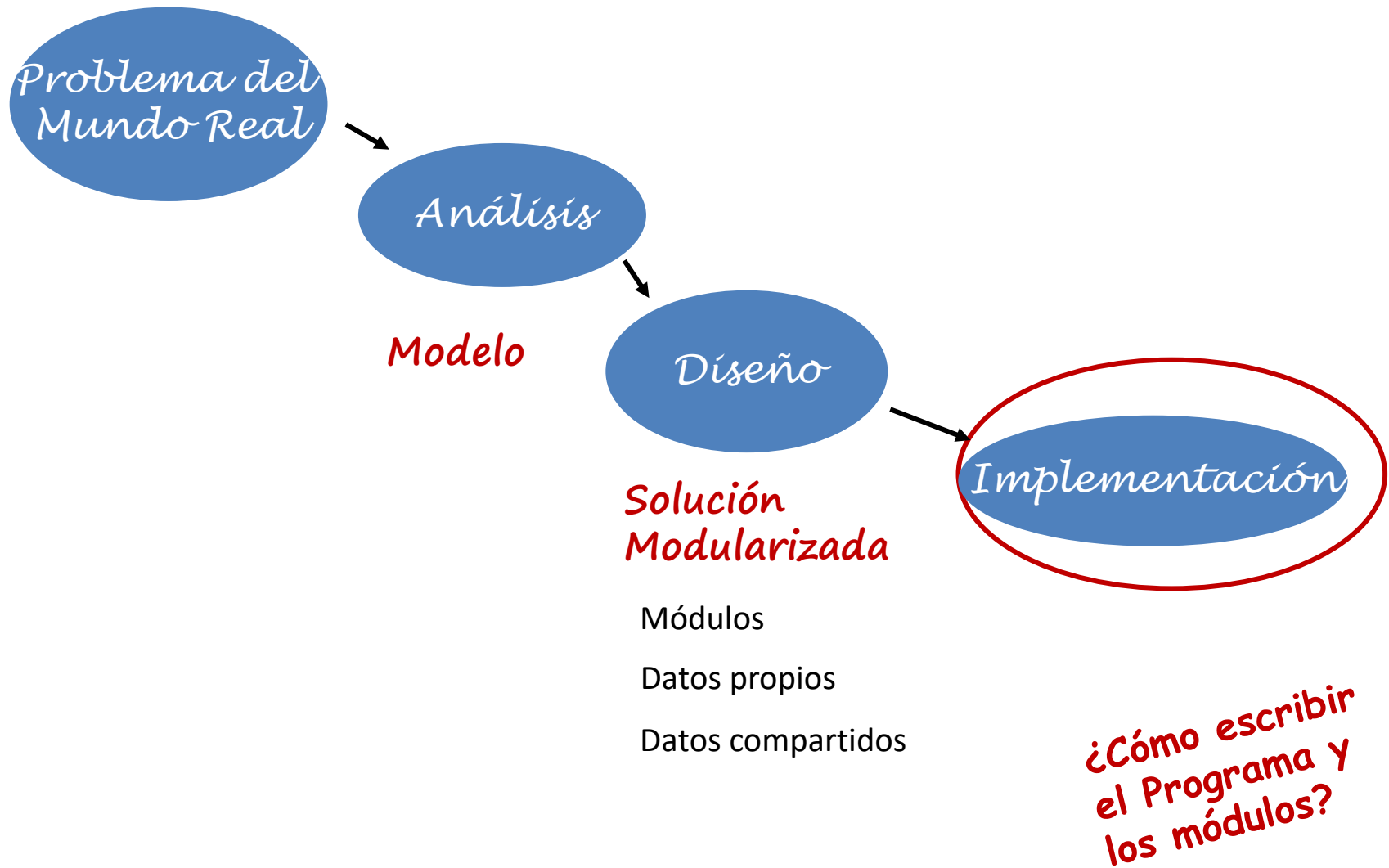
Como resultado de la etapa de Diseño se tiene:



- Cuales son los módulos
- Cual es el objetivo de cada uno
- Cuales son los datos propios
- Cuales son los datos compartidos con otros módulos.
- Cuales es el conjunto de acciones para alcanzar ese objetivo

Esta etapa no depende del lenguaje de programación que se use...

Avanzamos a la etapa de Implementación



2 Procedimientos y Funciones

- Se debe elegir el lenguaje de programación para escribir los algoritmos de cada módulo y la declaración de sus datos
- Los lenguajes de programación ofrecen diversas opciones para implementar la modularización.

Definición del módulo

¿Qué hace el módulo cuando se ejecuta?



- Encabezamiento (Interface)
 - Tipo de módulo
 - Identificación
 - Datos de comunicación
- Declaración de tipos
- Declaración de variables
- Sección de instrucciones ejecutables

Invocación del módulo

¿Qué se hace cuando se quiere usar el módulo?



- Se debe conocer de qué manera se invoca al módulo para que ejecute sus acciones

La invocación
puede hacerse
mas de una
vez

¿Qué ocurre
con el flujo de
control del
programa?

¿Qué tipos de módulos ofrece Pascal?

PROCEDIMIENTOS
(PROCEDURE)

FUNCIONES
(FUNCTION)

Tienen características comunes, pero ciertas particularidades determinan cual es el mas adecuado para implementar un módulo particular

- ¿El módulo devuelve datos?
 - ¿Cuántos datos devuelve?
 - ¿De qué tipo son los datos que devuelve?
 - ¿Qué tipo de acciones ejecuta el módulo?

¿PROCEDURE?

¿FUNCTION?

¿Cuáles son los aspectos que los diferencian?

- Encabezamiento del módulo
- Invocación
- Lugar donde retorna el flujo de control una vez ejecutado el módulo

PROCEDURE

Conjunto de instrucciones que realiza una tarea específica y como resultado puede retornar 0, 1 o más valores.

¿Cómo se define el módulo?

```
Procedure nombre (lista de parámetros formales);
```

```
Type
```

```
.....
```

```
Var
```

```
.....
```

```
begin
```

```
.....
```

```
.....
```

```
end;
```

Encabezamiento

Declaración de tipos internos del módulo (opcional)

Declaración de variables internas del módulo (opcional)

Sección de instrucciones

PROCEDURE

Conjunto de instrucciones que realiza una tarea específica y como resultado puede retornar 0, 1 o más valores.

¿Cómo se invoca el módulo?

```
Program uno;
```

```
.....
```

```
procedure Calculo (Parámetros Formales);
```

```
Type
```

```
... *
```

```
Var
```

```
.....
```

```
Begin
```

```
.....
```

```
... *
```

```
End;
```

```
Begin
```

```
.....
```

```
Calculo (parámetros actuales);
```

```
.....
```

```
End.
```

Conjunto de instrucciones que realiza una tarea específica y como resultado puede retornar 0, 1 o más valores.

Luego de ejecutado el módulo ¿Qué instrucción se ejecuta?

¿Qué ocurre
con el flujo de
control del
programa?

Luego de ejecutado el módulo, el flujo de control retorna a la instrucción siguiente a la invocación del módulo

Program uno;

.....

procedure Calculo (Parámetros Formales);

Type

... .

Var

.....

Begin

.....

... .

End;

Begin

.....

Calculo (parámetros actuales);

.....

End.



PROCEDURE

Puede ocurrir que un módulo Procedure contenga además, de la declaración de tipos y variables propias, la definición de otros módulos. Se dice que el procedimiento contiene módulos anidados. Por ejemplo:

```
Procedure principal (lista de parametros formales);
```

```
Type
```

```
.....
```

```
Var
```

```
.....
```

```
Procedure uno (lista de parametros formales);
```

```
begin
```

```
end;
```

```
Procedure dos (lista de parametros formales);
```

```
begin
```

```
end;
```

```
Var .....
```

```
Begin {instrucciones ejecutables del procedure principal}
```

```
...
```

```
  uno (parametros actuales);
```

```
  dos (parametros actuales);
```

```
.....
```

```
end;
```

FUNCTION

Conjunto de instrucciones que realiza una tarea específica y como resultado retorna un único valor de tipo simple.

¿Cómo se define el módulo?

```
Function nombre (lista de parámetros formales): tipo;
```

Encabezamiento

```
Type .....
```

Declaración de tipos
(opcional)

```
Var .....
```

Declaración de variables
(opcional)

```
begin
```

```
.....
```

```
nombre := ...;
```

```
end;
```

— **Asignación
(obligatoria)**

Sección de instrucciones

FUNCTION

Conjunto de instrucciones que realiza una tarea específica y como resultado retorna un único valor de tipo simple.

¿Cómo se invoca el módulo?

```
Program otro;
```

```
Function cubo(Parámetros Formales):integer;  
Type  
    ... *  
Var  
    .....  
Begin  
    .....  
    ... *  
End;
```

```
Begin  
    ... *  
    ... *  
    write (cubo (parametro actual));  
    ... *  
End.
```

```
Program otro;
```

```
Function cubo(Parámetros Formales):integer;  
Type  
    ... *  
Var  
    .....  
Begin  
    .....  
    ... *  
End;
```

```
Var a: integer;  
begin  
    ... *  
    ... *  
    a := cubo (parametro actual));  
    ... *  
End.
```

¿Qué ocurre con
el flujo de control
del programa?

Luego de ejecutado el módulo, el flujo de control retorna a la misma instrucción de invocación del módulo

FUNCTION

Conjunto de instrucciones que realiza una tarea específica y como resultado retorna un único valor de tipo simple.

¿Cómo se invoca el módulo?

```
Program otro;
```

```
Function cubo(Parámetros Formales):integer;  
Type  
    ... *  
Var  
    .....  
Begin  
    .....  
    ... *  
End;
```

```
begin  
    ... *  
    ... *  
    if (cubo (parametro actual) > 100) then ....;  
End.
```

```
Program otro;
```

```
Function cubo(Parámetros Formales):integer;  
Type  
    ... *  
Var  
    .....  
Begin  
    .....  
    ... *  
End;
```

```
Var a: integer;
```

```
begin....  
    ... *  
    ... *  
    while (cubo (parametro actual) > 50) do  
        ..... *  
End.
```

También puede ocurrir que un módulo function contenga además, de la declaración de tipos y variables propias, la definición de otros módulos procedimientos y/o funciones.

Program Ejemplo8;

*¿Qué
imprime el
programa?*

```
Function EsPar (num:integer): boolean;  
  var resto:integer;  
  begin  
    resto := num mod 2;  
    if resto = 0 then EsPar := true  
      else EsPar:= False;  
  end;
```

```
var x: integer;  
begin  
  read (x);  
  if EsPar(x) then writeln ('El número ', x, 'es par')  
    else writeln ('El número ', x, 'no es par')  
end.
```

Esquema general de un programa que utiliza módulos

program uno;

Const

{Declaración de constantes del programa}

Type

{Declaración de tipos definidos}

Var

{variables}

Procedure Calculo (parámetros formales);

.....

Function EsPar (parámetros formales): Boolean;

...

Var

{variables}

Begin *{Sección del programa principal}*

.....

End.

Zona de Declaración de constantes

Zona de Declaración de tipos

Zona de Declaración de variables accedidas desde la sección del programa y los módulos

Zona de Declaración de los módulos

Zona de Declaración de variables accedidas solo desde la sección del programa

Zona de Instrucciones ejecutables

3

Comunicación entre módulos

Variables globales

Parámetros

Analicemos el alcance de Variables...

```
Program dos;
```

```
Var
```

```
  a, b: integer;
```

Variables
globales

```
procedure calculo(parámetros formales);
```

```
  var
```

```
    x: integer;
```

Var b:integer;

```
  Begin
```

```
    x:= 9; a:= 100; b:=0;
```

```
    write (x);
```

```
  End;
```

Parámetros

¿Dónde se pueden utilizar a y b?

¿Dónde se puede utilizar x?

¿Dónde se puede utilizar h?

¿Qué pasa si dentro de calculo se declara b: integer?

¿Qué pasa si dentro de calculo se declara b: char?

```
  Var
```

```
    h: char;
```

```
  Begin
```

```
    a:= 80;
```

```
    b:= a * 2;
```

```
    h:= 'A';
```

```
    calculo(parámetros actuales);
```

```
  End.
```

Variable
Local del
programa

Variables globales, locales y parámetros

→ **Variable global:** su declaración se hace en la sección de declaración del programa principal, es decir fuera de todos los módulos del programa y podrá ser usada en el programa y en todos los módulos del mismo. Por lo tanto, podrían ser utilizadas para la comunicación entre el programa y los módulos.

→ **Variable local al módulo:** su declaración se hace en un módulo particular y sólo podrá ser usada por ese módulo. Si este módulo contiene a su vez otros módulos, entonces esa variable podría ser también usada por todos los módulos interiores, si está declarada previo a ellos.

→ **Variable local al programa:** su declaración se hace antes de la sección de instrucciones ejecutables del programa y después de la declaración de los módulos del programa. Su uso se limita a la sección de instrucciones ejecutables.

→ **Parámetros:** son los datos que se utilizarán para la comunicación entre el programa y los módulos, de una manera explícita.

Analicemos el alcance de variables en el siguiente ejemplo...

Alcance de los datos

```
program ejemplo;  
var a, b: integer;
```

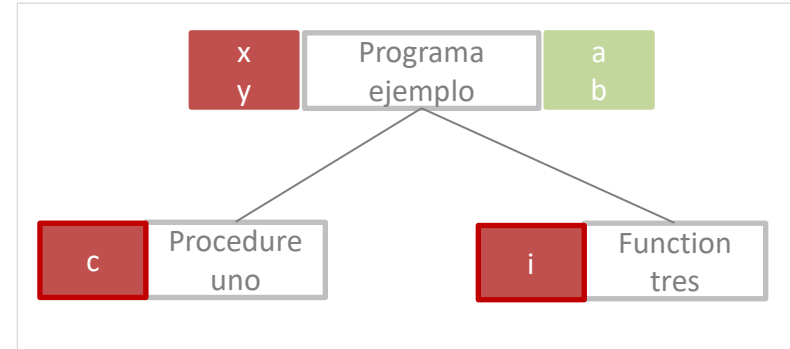
```
procedure uno(parámetros formales);  
var c: char;  
begin  
  b := a * 2;  
  c:= 'A';  
  writeln(a, b, c);  
  a:= tres (parámetros actuales);  
end;
```

```
Function tres (parámetros formales): tipo;  
var i: integer;  
begin  
  i:= a+b;  
  uno (parámetros actuales);  
  tres:= (c+i)/100;  
end;
```

```
Var x, y: integer;  
begin  
  x:= 5; y:= 20 mod 10;  
  uno (parámetros actuales);  
  a:= tres (parámetros actuales);  
  writeln(x, y);  
  writeln(a, b);  
end.
```

Supongamos que identificamos:

- variables locales del módulo
- variables “globales”



Teniendo en cuenta el alcance de las variables y la visibilidad de los módulos, analizar:

¿Qué invocaciones son válidas?

¿Es correcta la utilización de las variables en el programa principal?

¿Y en el procedure uno?

¿Y en la function tres?

Alcance de los datos

```
program otroejemplo;  
var a, b: integer;
```

```
procedure cuatro (parámetros formales);  
var c: char;  
procedure cinco (parámetros formales);  
var d: real;  
begin  
  b := a * 2 + 4; d:= 2,5;  
  writeln(a, b, c, d);  
end;
```

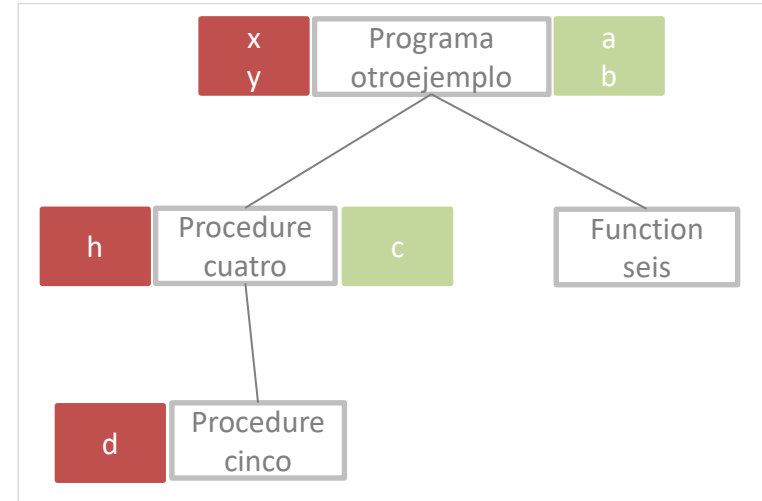
```
var h: integer;  
begin  
  b := a * 2;  
  cinco (parámetros actuales);  
  c:= 'A';  
  writeln(a, b, c, h);  
  h:= seis (parámetros actuales);  
end;
```

```
Function seis (parámetros formales): tipo;  
begin  
  a:= a+b;  
  seis:= (a+c)/100;  
end;
```

```
Var x, y: integer;  
begin  
  x:= 5; y:= 20 mod 10;  
  cuatro (parámetros actuales);  
  cinco (parámetros actuales);  
  a:= seis (parámetros actuales);  
  writeln(x, y);  
  writeln(a, b);  
end.
```

Supongamos que identificamos:

- variables locales del módulo
- variables “globales”



Teniendo en cuenta el alcance de las variables y la visibilidad de los módulos, analizar:

¿Qué invocaciones son válidas?

¿Es correcta la utilización de las variables en el programa principal?

¿Y en el procedure cuatro?

¿Y en el procedure cinco?

¿Y en la function seis?

Alcance de los datos

```
Program dos;  
Var  
  a, b: integer;
```

Variables
globales

```
procedure calculo;  
  var x: integer;  
  Begin  
    x:= 9; a:= 100;  
    write (x);  
  End;
```

Variable
Local del módulo

```
procedure mostrar;  
  var y: char;  
  Begin  
    y:= 'P';  
    a:= a+10;  
    write (a);  
    write (b);  
  End;
```

Variable
Local del módulo

```
Var  
  h: char;  
Begin  
  a:= 80;  
  b:= a * 2;  
  h:= 'A';  
  calculo;  
  mostrar;  
  write (a);  
End.
```

Variable Local del
programa

Datos propios



Variables locales
(al módulo y al programa)

Datos compartidos



¿Variables globales?

Desventajas
de la
comunicación
a través de
variables
globales

- Posibilidad de perder integridad de los datos, al modificar involuntariamente en un módulo alguna variable que luego deberá utilizar otro módulo. ¿Independencia? ¿Reusabilidad?
- Dificultad durante la etapa de la verificación
- Demasiadas variables en la sección de declaración
- Posibilidad de conflicto entre los nombres de las variables utilizadas por diferentes programadores
- Falta de especificación del tipo de comunicación entre los módulos
- Uso de memoria

No se recomienda el uso de variables globales

Se recomienda una solución que combina:

OCULTAMIENTO DE DATOS Y PARAMETROS

▪ El **ocultamiento de datos** significa que los datos exclusivos de un módulo o programa **NO** deben ser "visibles" o utilizables por los demás módulos.

Variables locales del módulo



Los datos propios del módulo se declararan locales al módulo

Variables locales del programa



Los datos propios del programa se declararan locales al programa

▪ El **uso de parámetros** significa que los datos compartidos se deben especificar como parámetros que se trasmiten entre módulos.



Los datos compartidos se declararán como parámetros.

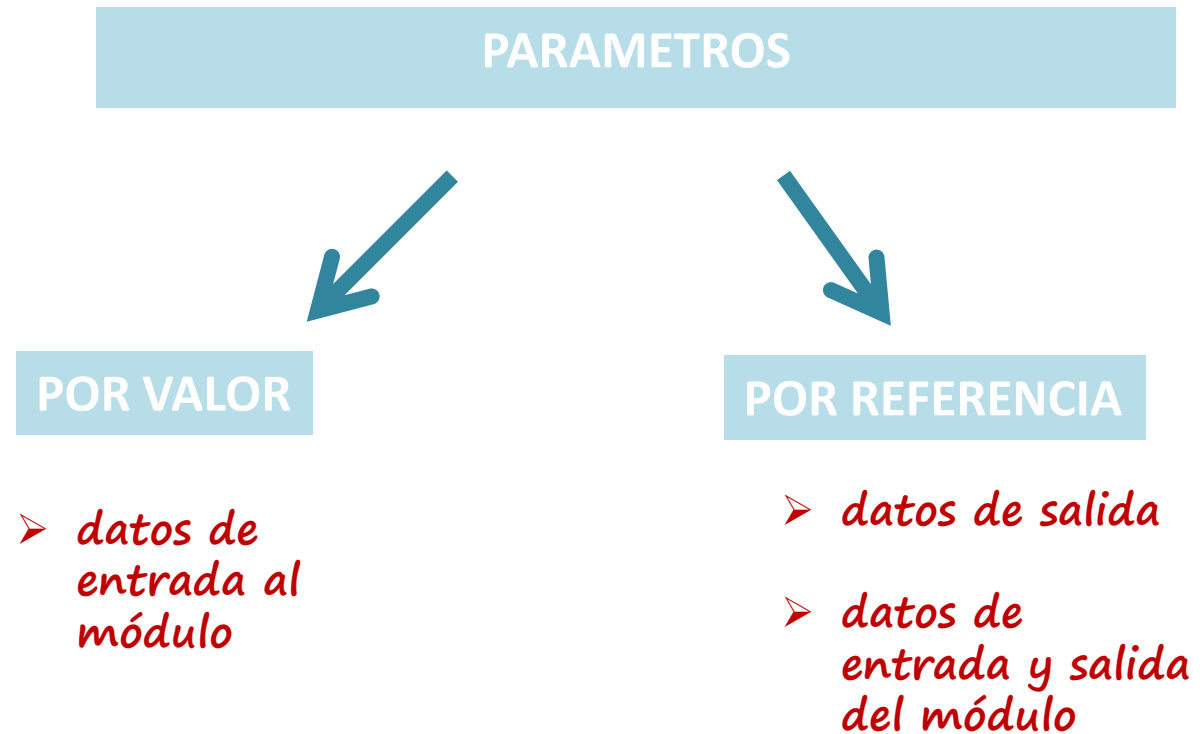
No nos sirven para la comunicación!!

El parámetro se define como una variable que representa un dato compartido entre módulos o entre un módulo y el programa principal.

Por lo tanto, el dato compartido se especificará como un parámetro que se transmite entre los módulos.

Del curso de Ingreso se sabe que el dato compartido puede ser un dato de entrada al módulo, un dato de salida del módulo o bien un dato de entrada y salida del módulo .

¿Cómo se implementan los parámetros en Pascal?



Un parámetro por valor es un dato de entrada que significa que el módulo recibe una copia de un valor proveniente de otro módulo o del programa principal.

Con este dato el módulo puede realizar operaciones y/o cálculos, pero fuera del módulo ese dato NO reflejará cambios.

Parámetro
por valor

```
Program ejemplo1;  
  
  Procedure uno (x:integer);  
  Begin  
    x:= x+1;  
    write (x);  
  End;  
  
  var num: integer;  
  
  Begin  
    num:=9;  
    uno (num);  
    write (num);  
  End.
```

¿Qué valores
imprime?

ejemplo1

num=9

Memoria

Un parámetro por referencia es un dato de salida o entrada/salida que contiene la dirección de memoria donde se encuentra la información compartida con otro módulo o programa que lo invoca.

El módulo que recibe este parámetro puede operar con la información que se encuentra en la dirección de memoria compartida y las modificaciones que se produzcan se reflejarán en los demás módulos que conocen esa dirección de memoria compartida.

Parámetro
por
referencia

```
Program ejemplo2;  
  
  Procedure dos (var x:integer);  
  Begin  
    x:= x+1;  
    write (x);  
  End;  
  
var num: integer;  
Begin  
  num:=9;  
  dos (num);  
  write (num);  
End.
```

¿Qué valores
imprime?

Ejemplo2
num=90



Memoria

```
Program ejemplo1;
```

```
Procedure uno (x:integer);
```

```
Begin
```

```
  x:= x+1;
```

```
  write (x); {1}
```

```
End;
```

```
var num: integer;
```

```
Begin
```

```
  num:=9;
```

```
  uno (num);
```

```
  write (num); {2}
```

```
End.
```

En {1} se muestra 10 como valor de x

En {2} se muestra 9 como valor de x

```
Program ejemplo2;
```

```
Procedure dos (var x:integer);
```

```
Begin
```

```
  x:= x+1;
```

```
  write (x); {3}
```

```
End;
```

```
var num: integer;
```

```
Begin
```

```
  num:=9;
```

```
  dos (num);
```

```
  write (num); {4}
```

```
End.
```

En {3} se muestra 10 como valor de x

En {4} se muestra 10 como valor de x

¿Cómo se explican los valores que se imprimen?

Consideraciones...

Un **parámetro por valor** debe ser tratado como una variable local del módulo.

La utilización de este tipo de parámetros puede significar una utilización importante de memoria.

Los **parámetros por referencia** en cambio operan directamente sobre la dirección de la variable original, en el contexto del módulo que llama.

Esto significa que no requiere memoria local.

Variables
globales
vs
Parámetros

Variables globales

- Posibilidad de perder integridad de los datos, al modificar involuntariamente en un módulo alguna variable que luego deberá utilizar otro módulo
- Posibilidad de conflicto entre los nombres de las variables utilizadas por diferentes programadores
- Dificultad durante la etapa de la verificación

Parámetros



Los módulos no pueden afectar involuntariamente los datos de otros módulos. ¡Independencia! ¡Reusabilidad!



Se reduce significativamente la cantidad de variables globales



Permite distinguir el tipo de comunicación



Los parámetros por referencia no utilizan memoria local



El uso de parámetros por valor puede significar una importante utilización de memoria local

COMUNICACIÓN DE DATOS ENTRE MODULOS Y PROGRAMA

```
graph TD; A[COMUNICACIÓN DE DATOS ENTRE MODULOS Y PROGRAMA] --> B[VARIABLES GLOBALES]; A --> C[PARAMETROS]; C --> D[Por valor]; C --> E[Por referencia];
```

VARIABLES
GLOBALES

PARAMETROS

Por valor

Por referencia

*No recomendables
para la
comunicación!!*

En resumen

Cuando se invoca a un módulo se deben tener cuenta las restricciones propias del lenguaje de implementación. En Pascal:

- La invocación al módulo (parámetros actuales)
- La interface del módulo (parámetros formales)
- Los parámetros actuales y formales



Deben coincidir en cantidad y tipo de dato

Se relacionan 1 a 1

Consideraciones
generales

```
Program ejemplo3;  
  Procedure Calcular (x, y: integer;  
                     var suma, prod: integer);  
  begin  
    suma:= x + y;  
    prod:= x * y;  
  end;
```

```
Var  val1, val2, s, p: integer;  
Begin
```

```
  Calcular (6, 17, s, p);
```



```
  val1:= 10; val2:= 5;  
  Calcular (val1, val2, s, p);
```



```
  Calcular (23, val2, 14, p);
```



```
End.
```

¿Qué imprime en cada sentencia write para la siguiente secuencia?
10 15 3 22 -5 24

Ejercitación

```
Program ejemplo7;
```

```
procedure calcular ( n: integer; var cant, aux: integer);  
Begin  
    cant := cant + 1;  
    if (n mod 2 = 0) then aux := aux+1;  
end;
```

```
Var
```

```
    a,c, num: integer;
```

```
begin
```

```
    writeln ('a= ',a, ' c= ', c);
```

```
    writeln;
```

```
    c:=0;
```

```
    a:=0;
```

```
    write ('Ingrese un numero (para finalizar 24): ');
```

```
    readln (num);
```

```
    while (num <> 24) do begin
```

```
        calcular (num,c, a);
```

```
        write ('Ingrese un numero (para finalizar 24): ');
```

```
        readln (num);
```

```
    end;
```

```
    writeln;
```

```
    writeln ('a= ',a, ' c= ', c);
```

```
    readln;
```

```
end.
```


Ejercitación

Program Ejemplo4;

```
procedure Imprimir (var a:integer; b: integer; c: integer);  
begin  
    writeln ('a= ', a, ' b= ', b, ' c= ', c);  
    writeln;  
    a := 10;  
    c := a + b;  
    b := b * 5;  
    writeln ('a= ', a, ' b= ', b, ' c= ', c);  
    writeln;  
end;
```

var x, y, z: integer;

begin

y := 6;

z := 5;

writeln ('x= ', x, ' y= ', y, ' z= ', z);

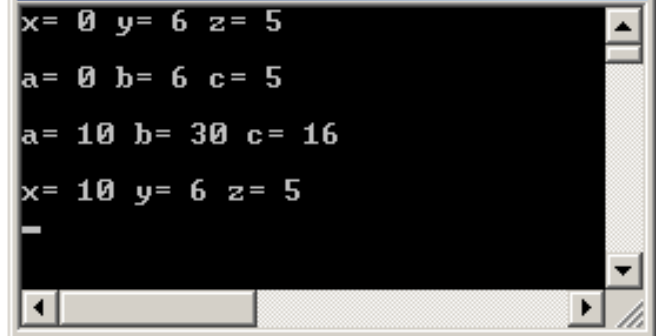
writeln;

imprimir (x, y, z);

writeln ('x= ', x, ' y= ', y, ' z= ', z);

readln;

end.



```
x= 0 y= 6 z= 5  
a= 0 b= 6 c= 5  
a= 10 b= 30 c= 16  
x= 10 y= 6 z= 5  
_
```