Modularización

Introducción a la Programación
Tecnicaturas Universitarias
Facultad de Informática
Univ.Nac. del Comahue

Indice

Modularización: Introducción

Modularizar es una estrategia de resolución de problemas y de ingeniería de software(*) que consiste en dividir el problema original en un conjunto de subproblemas.

(*) Ingeniería de software es la disciplina que estudia los principios y metodologías para el desarrollo y mantenimiento de sistemas software

Modularización: Introducción

Estrategia “Divide y Vencerás”

1) Dividir el problema en subproblemas más simples
2) Resolver los problemas simples
3) Resolver el problema original a partir de las soluciones de los problemas más simples

Modularización: Introducción

Estrategia “Divide y Vencerás”

Entonces…

Vamos a **crear módulos** para resolver los subProblemas y conseguir la solución al Problema!

Modularización: Introducción

Un **módulo** es un conjunto de instrucciones que pueden ser ejecutadas las veces que se requiera. El módulo puede ser llamado desde cualquier punto del programa.

Es una de las herramientas más importes en cualquier lenguaje de programación.

Modularización: Introducción

módulo

Entrada — **Parámetros**

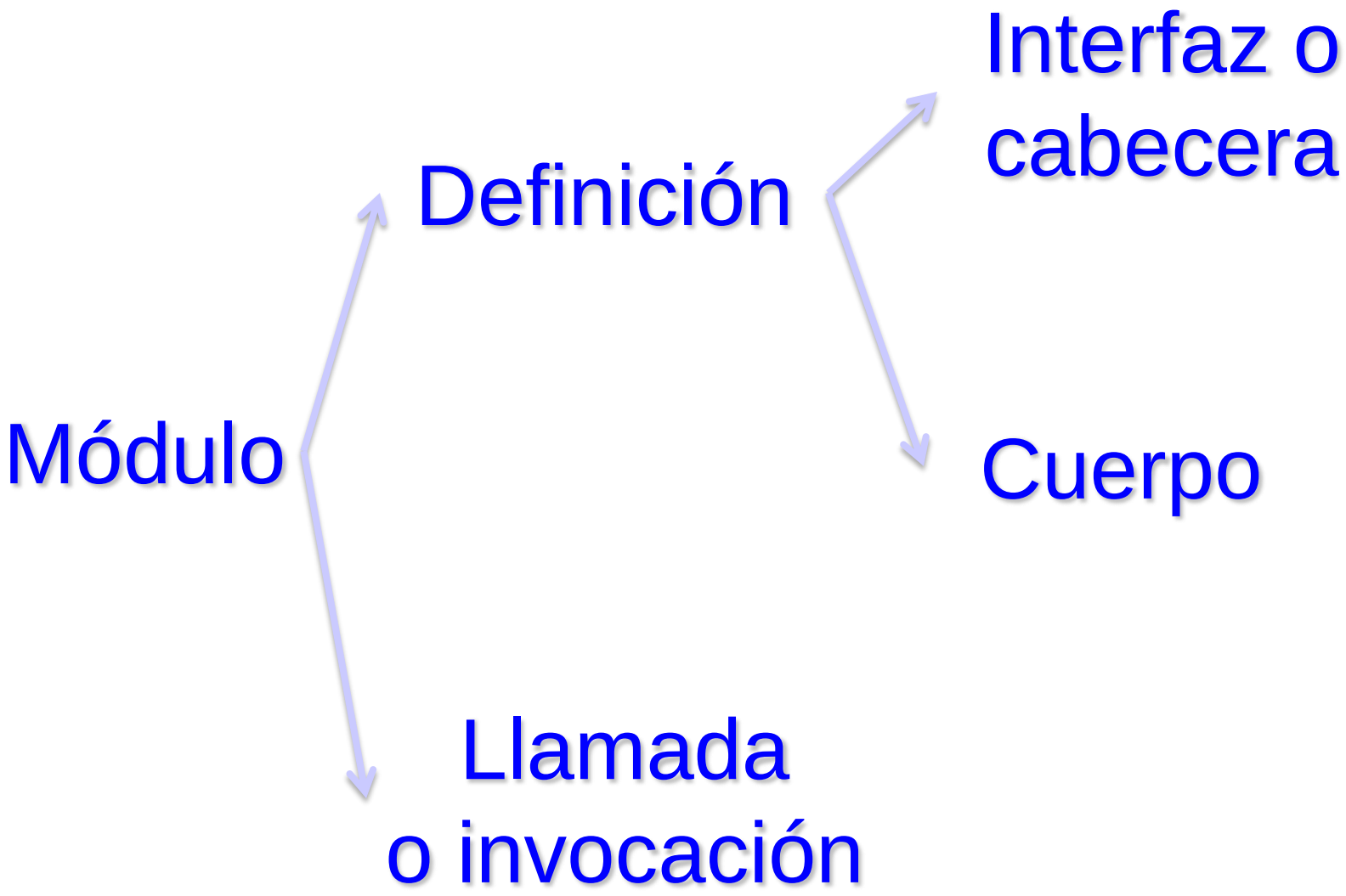
Proceso — **Conjunto de instrucciones**

Salida — **Valor de Retorno**

Modularización: Introducción

Dependiendo del Lenguaje de programación podemos referirnos a los ***módulos*** *como* ***subprogramas***, ***funciones*** *o* ***procedimientos***

Módularización



Módularización

Módulo

Definición

Interfaz o cabecera

¿Qué hacé? ¿Qué problema resuelve?
¿Qué necesita para hacer? (entrada)
¿cuál es el resultado? (retorno)
Es la parte **visible al invocar** el módulo

Cuerpo

¿Cómo lo hace?
Cómo resuelve el problema

Llamada o invocación

Utiliza la interfaz, es decir, utiliza la parte visible del módulo.
Provee los valores de entrada y espera el resultado

Módularización: analogía

Interfaz o cabecera

Definición

¿Qué hacé?: **Provee Agua para mate**
¿Qué necesita para hacer? (entrada) **Ficha**
¿cuál es el resultado? (retorno) **Agua Caliente**

Módulo

ExpenderAguaCaliente

Cuerpo

¿Cómo lo hace?

Llamada o invocación

Módularización

1) Definición del módulo. Especificamos el módulo estableciendo su **Nombre**, sus **Parámetros Formales o de Entrada**, **Retorno** y las **instrucciones del cuerpo**

> ***Parámetro Formal*: variable declarada en la cabecera del módulo. Almacena el valor de entrada cuando el módulo es invocado**

2) Llamada o Invocación del módulo. Hacemos uso del módulo definido. Se utiliza el **nombre** especificado en la **definición** y se establecen los valores de los **Parámetros Actuales** que serán asignados a los Parámetros Formales.

> ***Parámetro Actual*: Es una expresión (un literal, constante, variable, combinación de operandos y operadores) que se resuelve y su valor resultado se almacena en el parámetro formal.**

Módularización

Módulo

Definición

Interfaz o cabecera

- Documentar ¿qué hace?
- Nombre
- Parámetros formales o de entrada (0 a N variables)
- Tipo de dato del Retorno (0 o 1 dato de retorno)

Cuerpo

Variables internas +
Conjunto de instrucciones que procesan los parámetros formales y generan el retorno

Llamada o invocación

- utiliza el Nombre definido en la interfaz
- Parámetros Actuales (son expresiones *)

Coinciden en cantidad, orden y tipo de dato con los parámetros formales

* Literales, constantes, variables y combinaciones con operadores

1) Definición del módulo

Sintáxis

Definir una función en Pseudocódigo

(** Descripcion ¿qué hace el módulo? *)

MODULO nombre (tipo1 parametro1, …, tipoN parametroN) **RETORNO** tipoR / ∅

tipo var1,…,tipo varM

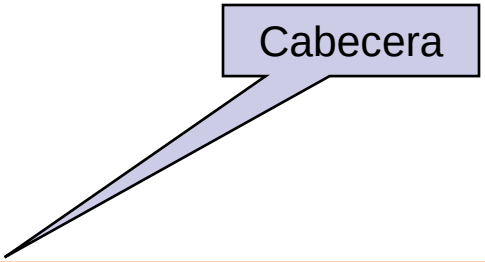
instruccion1

…

instruccionN

FIN MODULO

Definir una función en Pseudocódigo



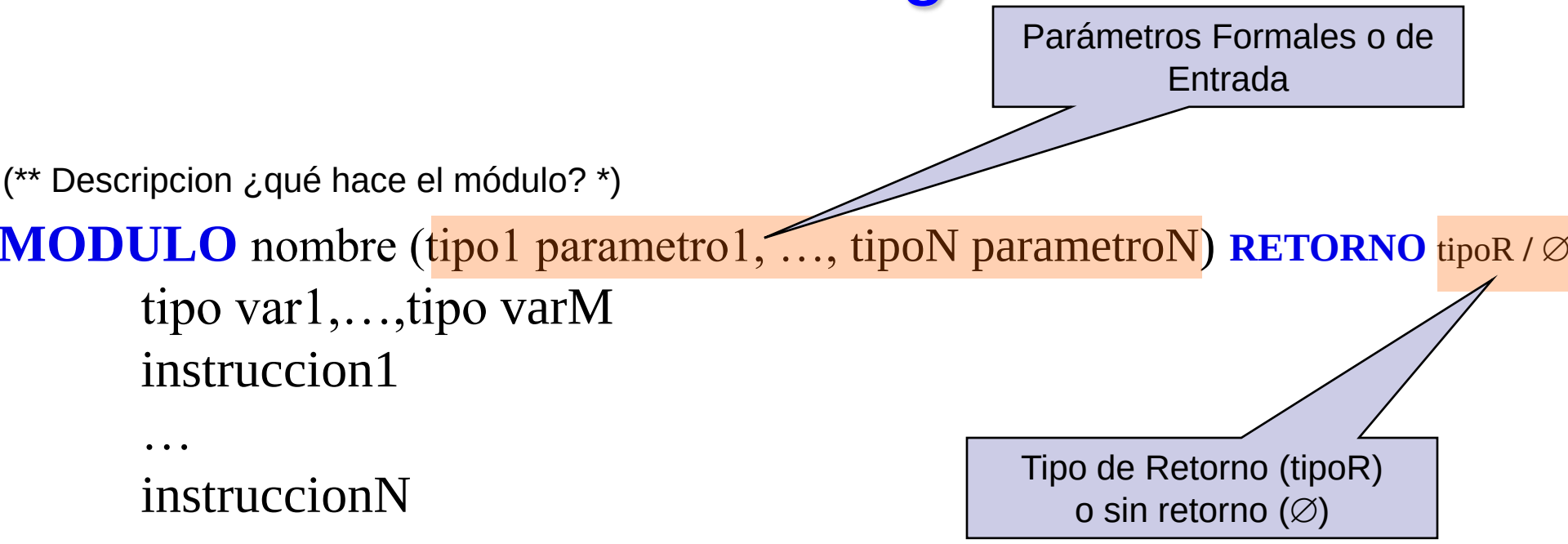
(** Descripcion ¿qué hace el módulo? *)

MODULO **nombre** (tipo1 parametro1, …, tipoN parametroN) **RETORNO** tipoR / ∅

tipo var1,…,tipo varM
instruccion1
…
instruccionN

FIN MODULO

Definir una función en Pseudocódigo



***Parámetro Formal*: variable declarada en la cabecera del módulo. Almacena el valor de entrada cuando el módulo es invocado**

Definir una función en Pseudocódigo

Documentación

Cabecera

(** Descripcion ¿qué hace el módulo? *)

MODULO nombre (tipo1 parametro1, …, tipoN parametroN) **RETORNO** tipoR / ∅

tipo var1,…,tipo varM
instruccion1
…
instruccionN

Cuerpo:
Declaración de variables internas + conjunto de instrucciones que procesan los parámetros formales o de entrada y resuelven la clase de problemas para el que se diseñó el módulo.

FIN MODULO

Documentación

- Cada función implementada por un programador realiza una tarea especifica.
- Cuando la cantidad de funciones disponibles para ser utilizadas es grande, puede ser difícil saber exactamente que hace una función.

Entonces **documentar**:

- **el propósito de la función**
- **el propósito de cada parámetro y su tipo de dato**
- **el tipo de dato retornado**

Definir una función en PHP

```
/**
* Descripcion ¿qué hace la función?
*@param tipo1 $parametro1
* ..
*@param tipoN $parametroN
*∅ / @return tipoR
*/
function nombre ($parametro1, …, $parametroN)
{
        /*tipo $var1,…,tipo $varM*/
        instruccion1;
        …
        instruccionN;

}
```

Palabra Reservada

Llaves para delimitar el cuerpo de la función

Definir una función en PHP

```
/**
* Descripcion ¿qué hace la función?
*@param tipo1 $parametro1
* ..
*@param tipoN $parametroN
*∅ / @return tipoR
*/
function nombre ($parametro1, …, $parametroN)
{
    /*tipo $var1,…,tipo $varM*/
    instruccion1;
    …
    instruccionN;

}
```

Etiquetas o tag

Documentación

Si la función no tiene retorno, no se escribe este tag.

Cabecera de la función

Parámetros Formales o de Entrada

Cuerpo de la Función:

conjunto de instrucciones que procesan los parámetros de entrada y resuelven la clase de problemas para el que se diseñó la función

Definir una función

Pseudocódigo:

```
(** Documentación de la función: descripción *)
MODULO nombre (tipo1 parametro1,…, tipoN parametroN) RETORNO tipoR / ∅
        Declaración de variables internas del módulo
        instruccion1
        …
        instruccionN
FIN MODULO
```

Traducción a PHP:

```
/** Descripcion ¿qué hace la función?
*@param  tipo1 $parametro1
* ..
*@param  tipoN $parametroN
*∅  / @return tipoR
 */
function nombre ($parametro1,…, $parametroN) {
        /*Declaración de variables internas de la función*/
        instruccion1;
        …
        instruccionN;
}
```

Definir una función

• Cada módulo **debe tener un nombre significativo** que lo **identifique**. (utilizaremos notación lowerCamelCase para el identificador del nombre al igual que las variables)

• Puede tener 0,1 o N Parámetros Formales o de entrada.

• Puede tener uno o ningún Retorno. (si tiene retorno por buena práctica de programación la instrucción de retorno se escribe al final del cuerpo de instrucciones (ver diapositiva 25))

Definir una función

Tipo de dato que retorna.
Si no retorna valor,
usamos el símbolo ∅

MODULO nombre (tipo1 parametro1, …, tipoN parametroN)**RETORNO** tipoR / ∅

    tipo var1,…,tipo varM
    instruccion1
    …
    instruccionN

FIN MODULO

?

Diseño: Definir una función CON retorno

Tipo de dato que retorna.

MODULO nombre (tipo1 parametro1, …, tipoN parametroN) RETORNO tipoR

tipo var1,…,tipo varM
instruccion1
…
instruccionN
RETORNO (E)

Instrucción de Retorno.
El intérprete resuelve la expresión E y retorna el valor
Como buena práctica, **siempre habrá una sola instrucción de RETORNO al finalizar el módulo**

FIN MODULO

la función definida retorna UN valor.
En Pseudocodigo: RETORNO
En PHP: return

Diseño: Definir una función SIN retorno

No tiene retorno

MODULO nombre (tipo1 parametro1, …, tipoN parametroN) RETORNO ∅

```
tipo var1,…,tipo varM
instruccion1
…
instruccionN
```

En el cuerpo de la función no existe instrucción RETORNAR

FIN MODULO

la función definida No retorna Valor.

Ejemplo “Importancia de los Parámetros”

Necesitamos una función que escriba en pantalla un saludo dirigido a una persona

Ejemplo "Importancia de los Parámetros"

Diseño

```
/** Muestra un saludo para Marta */
MODULO holaMarta( ) retorno ∅
        ESCRIBIR('Hola, Marta!')
        ESCRIBIR('Estoy programando en PHP!')
FIN MODULO
```

PHP

```php
/**
 * Muestra un saludo para Marta
 */
function holaMarta(){
    echo ("\n Hola, Marta!");
    echo ("\n Estoy Programando en PHP!");
}
```

Salida/ Pantalla

```
Hola, Marta!
Estoy programando en PHP!
```

Ejemplo "Importancia de los Parámetros"

Diseño

```
/** Muestra un saludo para Pablo */
MODULO holaPablo( ) retorno ∅
        ESCRIBIR('Hola, Pablo!')
        ESCRIBIR('Estoy programando en PHP!')
FIN MODULO
```

PHP

```php
/**
 * Muestra un saludo para Pablo
 */
function holaMarta(){

    echo ("\n Hola, Pablo!");
    echo ("\n Estoy Programando en PHP!");

}
```

Salida/
Pantalla

```
Hola, Pablo!
Estoy programando en PHP!
```

Ejemplo "Importancia de los Parámetros"

¿Vamos a crear una función por cada persona que tengamos que saludar?

… mejor especificaremos una función con **parámetros** (variables de entrada), que podamos **reusar** para cualquier nombre de persona …

Ejemplo "Importancia de los Parámetros"

Definición Módulo En Pseud.

```
/** Muestra un saludo dirigido a alguien */
MODULO hola( string alguien ) retorno ∅
        ESCRIBIR('Hola, ', alguien, '!')
        ESCRIBIR('Estoy programando en PHP!')
FIN MODULO
```

Parámetrizamos a quién vamos a saludar

Definición función en PHP

Invocación

```php
/**
 * Muestra un saludo dirigido a alguien
 * @param string $alguien
 */
function hola($alguien){
    echo ("\n Hola, ". $alguien ."!");
    echo ("\n Estoy Programando en PHP!");
}
hola("Marta");
hola("Pablo");
hola("Adrian");
```

Invocación con el valor de concreto a quién saludar.

Salida/ Pantalla

```
Hola, Marta!
Estoy Programando en PHP!
Hola, Pablo!
Estoy Programando en PHP!
Hola, Adrian!
Estoy Programando en PHP!
```

Repasando Parámetros

1 Definimos una función

2 Llamamos o Invocamos a la función definida

Parámetros Formales o de entrada

(son variables)

Coinciden en cantidad, orden y tipo (pero no, necesariamente, en nombre) con los Parámetros Formales

Parámetros Actuales

(variables, literales, expresiones, otras funciones con retorno)

Mostrando un resultado

- Las funciones que definimos hasta ahora muestran mensajes pantalla y no tienen retorno
- **Conviene** definir **funciones que se comporten como las funciones matemáticas** que se usan para hacer cálculos a partir de la entrada (parámetros), devolver resultados (retorno), y que puedan ser usados en otras expresiones.

LA MAYORÍA DE LAS FUNCIONES QUE DISEÑAREMOS NO ESCRIBIRÁN EN PANTALLA SINO QUE TENDRÁN RETORNO!!!

Importante: Funciones que Retornan Resultados

Contar con funciones es de gran utilidad, porque permite ir armando una biblioteca de funciones que resuelven problemas y que se pueden reutilizar para resolver nuevos problemas.

Más útil que tener una biblioteca donde los resultados se imprimen por pantalla, es contar con una biblioteca donde las **funciones retornan resultados**. El objetivo es utilizar dichas funciones y manipular los resultados a voluntad: imprimirlos, usarlos en cálculos más complejos, etc.

Funciones predefinidas en PHP: Funciones que Retornan Resultados

http://php.net/manual/es/ref.math.php

Funciones Matemáticas

- abs — Valor absoluto
- acos — Arco coseno
- acosh — Arco coseno hiperbólico
- asin — Arco seno
- asinh — Arco seno hiperbólico
- atan2 — Arco tangente de dos variables
- atan — Arco tangente
- atanh — Arco tangente hiperbólica
- base_convert — Convertir un número entre bases arbitrarias
- bindec — Binario a decimal
- ceil — Redondear fracciones hacia arriba
- cos — Coseno
- cosh — Coseno hiperbólico
- decbin — Decimal a binario
- dechex — Decimal a hexadecimal
- decoct — Decimal a octal
- deg2rad — Convierte el número en grados a su equivalente en radianes
- exp — Calcula la exponencial de e
- expm1 — Devuelve exp(numero)-1
  numero se aproxima a cero.
- floor — Redondear fracciones hacia abajo
- fmod — Devuelve el resto en punto flotante (módulo) de la división de los argumentos

abs

abs — Valor absoluto

```
number abs ( mixed $number )
```

Devuelve el valor absoluto de **number**.

Parámetros

number Valor decimal a convertir

Valores devueltos

El valor absoluto de **number**.

Ejemplo:

Programa Fuente (especificación):

```php
<?php
 $num1 = -5;
 $num2 = 10;
 $resultado = abs($num1) + abs($num2) +  5;
 echo "El resultado es: ". $resultado;
 echo "\nEl valor absoluto de ".$num1. " es ". abs($num1);
 echo "\nEl valor absoluto de ".$num2. " es ". abs($num2);
```

Ejecución:

```
El resultado es: 20
El valor absoluto de -5 es 5
El valor absoluto de 10 es 10
```

2) Invocación o llamada a la función

Llamada a una función definida sin retorno

Al no tener valor de retorno, la invocación no se puede: utilizar en una instrucción de escritura, tampoco se puede asignar a una variable, ni utilizar en una expresión (ver ejemplo diapositiva 31**)**

```
. . .
(*si la función no retorna valor, la llamo:*)
nombre( expresion1,expresion2, ..., expresionN )
. . .
```

Nombre de la función invocada

Lista de **parámetros Actuales** (la cantidad, el orden y el tipo de dato coincide con los parámetros Formales)

Expresion1, ..., expresionN puede ser un literal, una variable, una expresión o una función que retorna valor

Llamada a una función definida con retorno

Si la función llamada retorna un valor, podemos usarla dentro de otra instrucción. El intérprete resuelve la llamada a la función y utiliza el valor de retorno para seguir resolviendo la expresión

```
...
(*si la función retorna valor, la llamo dentro de otra expresión:*)
variable <-- nombre( expresion1,expresion2, ..., expresionN )
ESCRIBIR ("El resultado es ",  variable    )
...
```

Nombre de la función invocada

Lista de **Parámetros Actuales** (la cantidad, el orden y el tipo de dato coincide con los parámetros Formales)

Expresion1, ..., expresionN puede ser un literal, una variable, una expresióno una función que retorna valor

Ejecución de invocaciones a funciones

Código fuente

```
MODULO B
    instruccionB1
    instruccionB2
FIN MODULO

MODULO A
    instruccionA1
    instruccionA2
    B
    instruccionA4
FIN MODULO

PROGRAMA principal
    instruccion1
    instruccion2
    instruccion3
    A
    instruccion5
    B
FIN PROGRAMA
```

Invocación a **B**

Primera instrucción del código fuente a ser ejecutada

Invocación a **A**

Invocación a **B**

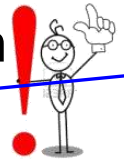
- **Siempre iniciamos la ejecución del programa principal**
- Cuando se invoca a un módulo el intérprete **transfiere el control** a dicho módulo. Una vez finalizada la última instrucción del módulo invocado **el control retornará** a la instrucción del programa o subprograma que lo llamó.

Ejecución de llamadas o invocaciones a funciones

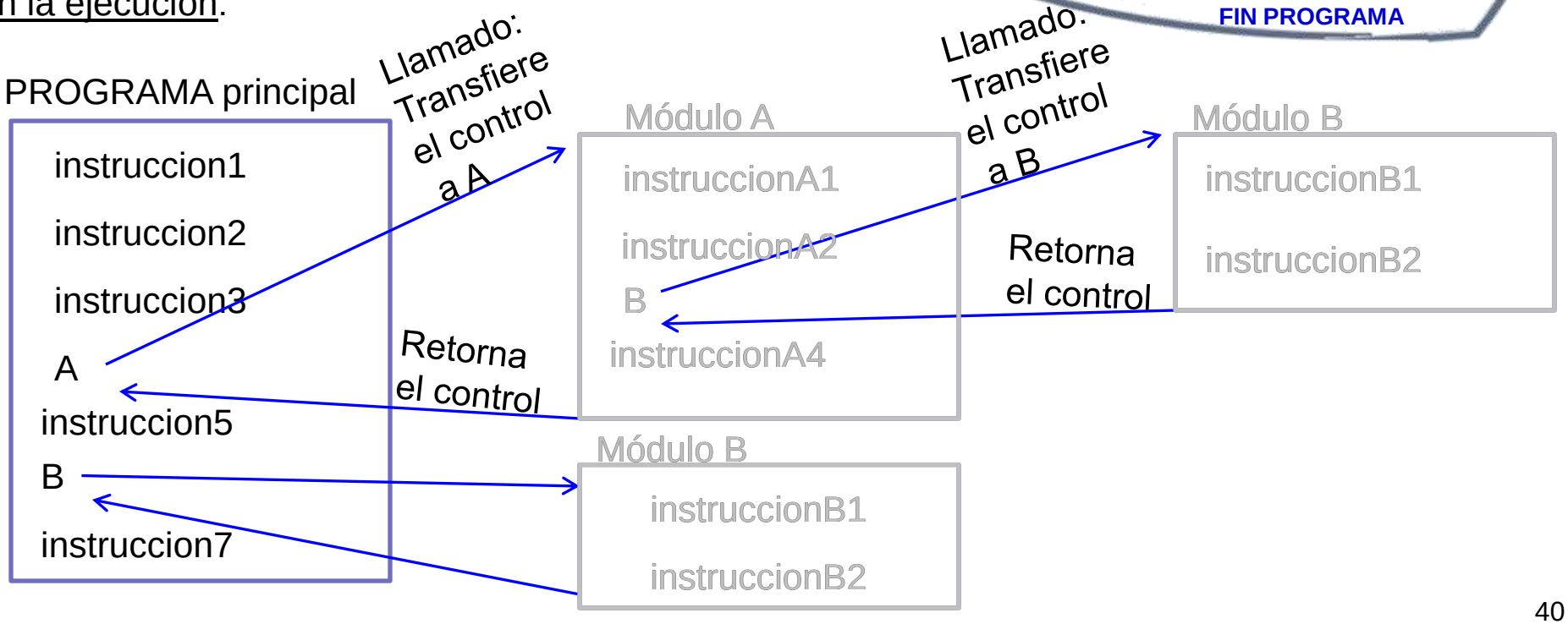
Código fuente

```
MODULO B
   instruccionB1
   instruccionB2
FIN MODULO
MODULO A
   instruccionA1
   instruccionA2
   B
   instruccionA4
FIN MODULO
PROGRAMA principal
   instruccion1
   instruccion2
   instruccion3
   A
   instruccion5
   B
   instruccion7
FIN PROGRAMA
```

- **Siempre iniciamos la ejecución del programa principal**



En la ejecución:



Traza

En la unidad 3 aprendimos a realizar la traza de un programa principal: construíamos una tabla cuyos nombres de columnas eran los nombres de variables + salida/pantalla.

Con la modularización, la traza se complejiza: armaremos 1 tabla para el programa principal (igual que la unidad 3) y tantas tablas como **invocaciones** funciones se ejecuten a partir del programa principal. **La tabla para la ejecución de una función tendrá una columna por cada parámetro formal + una columna por cada variable interna + retorno (si tiene) + salida/pantalla (si especifica instrucción escritura)**

La ejecución siempre comienza en el Programa Principal

Traza
en la materia Introducción a la Prog.

Lo más probable en un parcial es que se evalué una traza completa a partir de la ejecución del programa principal. (Test de integración)

Pero también puede suceder que se pida sólo la traza de una función en particular. Para lo cual se pedirá la ejecución de la función y se indicarán los valores de los parámetro actuales. (Test Unitario)

Modularización: Beneficios

Construcción:

- El lema **divide y vencerás** permite que los equipos de programadores trabajen en módulos independientes.

Modularización: Beneficios

Lectura

- **Aumenta la legibilidad y comprensión** de un programa.
- Un módulo debe ser entendible a partir de:
  - su nombre,
  - sus comentarios
  - los módulos que lo llaman

Modularización: Beneficios

- La modularización **aisla las modificaciones**, hace más manejable los cambios.
- La modularización **aisla errores**.
- El código de una operación aparecerá una sola vez, es decir, **reuso**.

Modularización: Requisitos

Para el uso apropiado de la modularización:

- Cada módulo debe hacer una única cosa (identificable a partir del nombre del módulo) y de forma generalizada
- Cada módulo oculta algo: importa **QUÉ** es lo que resuelve, no **CÓMO** lo resuelve.

Ejercicio:

Crear una función que calcule el cuadrado de un número.

Crear una función que a partir de los catetos calcule la hipotenusa de un triangulo rectángulo.

Luego crearemos un programa principal para solucionar el siguiente problema: Para ir desde mi casa hasta la de mi amigo, que vive en la misma manzana justo a la vuelta debo caminar una distancia X, luego doblar
90 grados y caminar otra distancia Y, cuántos metros caminaría si pudiera ir en línea recta desde mi casa hasta la de mi amigo?
Realice una traza