licenciatura en ciencias de la computación

programación I

<clase> 5 = subprogramas </clase>

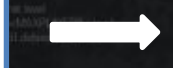
UNIVERSIDAD
NACIONAL DE CUYO

FACULTAD DE INGENIERIA
en acción continua...

Dra. Elina Pacini
Lic. Leandro Spadaro
Ing. Silvina Manganelli
Lic. Laura Noussan Lettry

programa

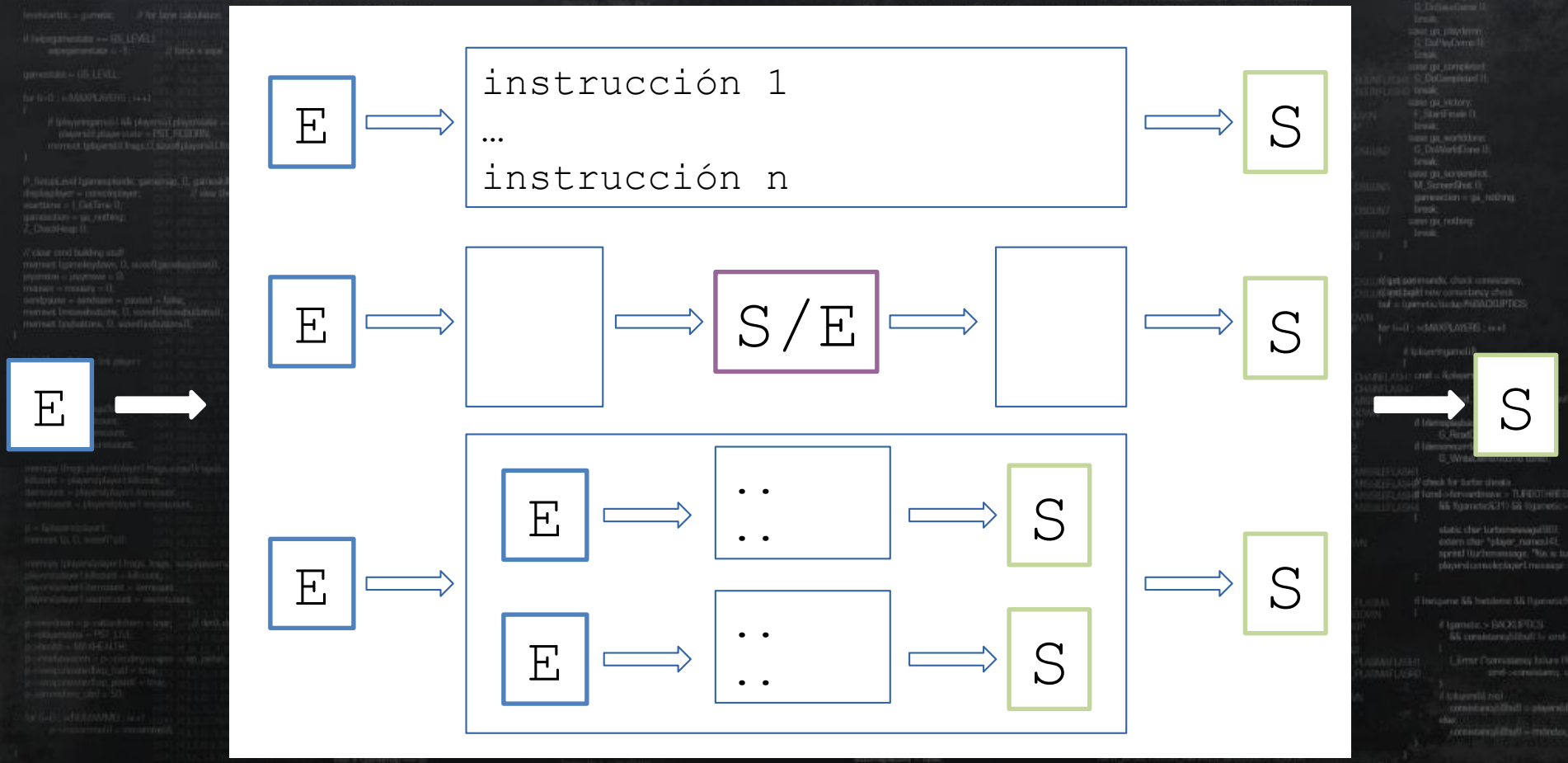
datos de entrada



```
instrucción 1

instrucción 2
…
…
…




…
instrucción N
```

datos de salida

subprogramas



¿es más fácil razonar un gran problema o un pequeño problema?
¿y si una parte de mi programa debe ser sustituido?
¿qué pasa si necesito ejecutar la misma tarea más de una vez?

estructura general

```
//calcula área de un círculo
Funcion area <- area_circulo(radio)
    Definir area Como Real
    area = 3.14 * radio * radio
FinFuncion

Algoritmo AreaConFunciones
    Definir r como Real
    Escribir "Ingrese un rádio: "
    leer r
    Escribir "El área del círculo es ", area_circulo(r)
FinAlgoritmo
```

nombre

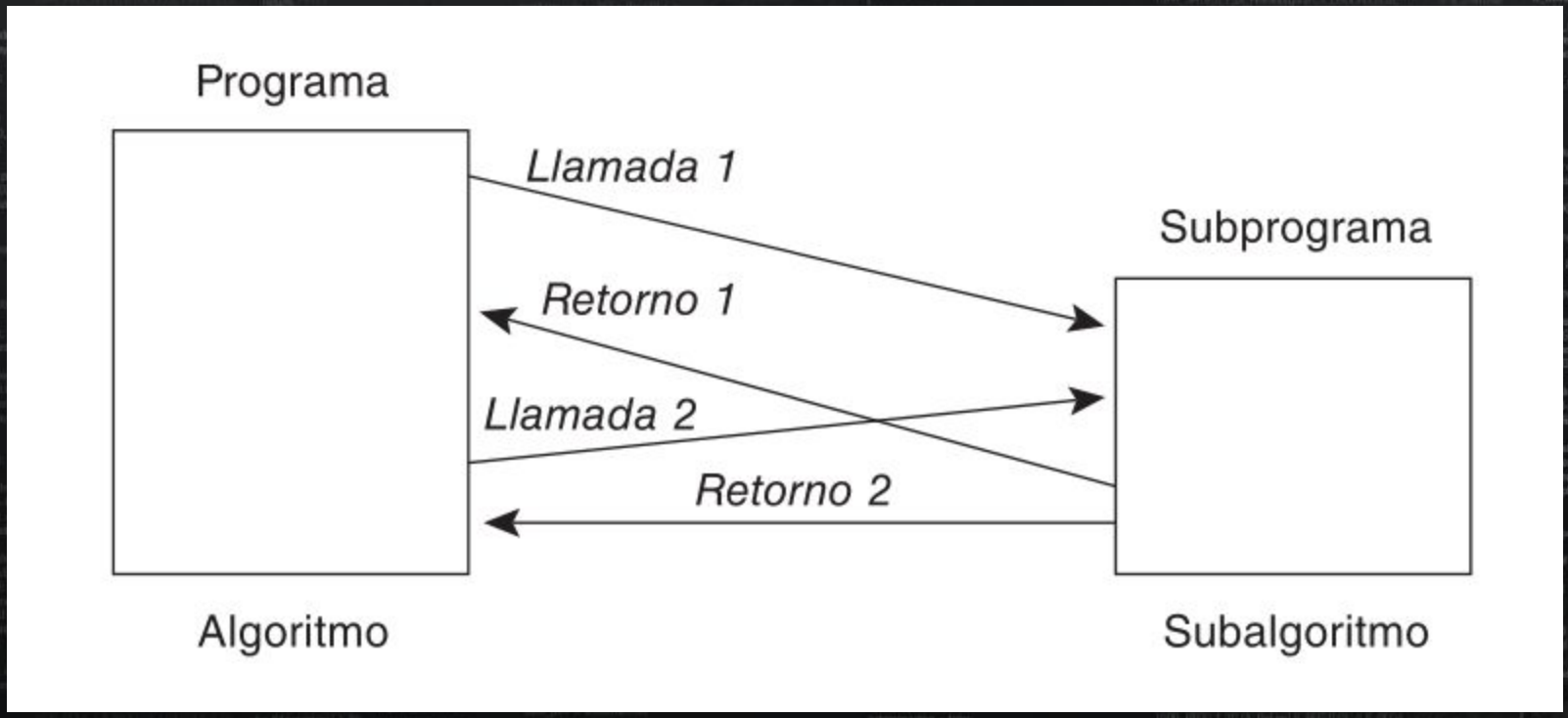
parámetros

acciones

valor de retorno*

documentación

flujo de control



tipos

función

procedimiento

(subprocesos)

¿qué los diferencia?

función: declaración

```
Funcion <variable_ret> <- <nombre_fun>(<parametros>)
  Definir <variable_ret> Como {tipo_dato}
  [declaraciones locales]
  [acciones] //cuerpo de la funcion
  //no olvidar dar valor a la variable de retorno!
  <variable_ret> = {expresión/valor_de_tipo_dato}
FinFuncion
```

```
(parámetro 1 {Por Valor|Por Referencia} [, parámetro 2]...)
```

¿cuántos valores retorna una función?

función: ej.

```
Funcion pot <- potencia(base, exponente)
    Definir pot Como Real
    Definir i como Entero
    pot = 1
    Para i = 1 Hasta abs(exponente)
      pot = pot * base
    Fin Para

    // si el exponente es negativo se invierte.
    Si exponente < 0 Entonces
      pot = 1/pot
    FinSi
FinFuncion
```

¿cuales son los parámetros de entrada y cuales de salida?

¿que se imaginan es ***abs()***?

```
...
leer b,e
p = potencia(b,e)
escribir "resultado ", p
...
```

procedimiento: declaración

```
SubProceso <nombre_proc>(<parametros>)
    [declaraciones locales]
    [acciones] //cuerpo del procedimiento
FinSubProceso
```

(parámetro 1 {Por Valor|Por Referencia} [, parámetro 2]...)

procedimiento: ej.

```
SubProceso division(ddo, div, cte Por Referencia, rto Por Referencia)
    cte = TRUNC(ddo / div)
    rto = ddo -(cte*div)        //rto = ddo MOD div
FinSubProceso

Algoritmo DivisionEntera
    Definir dividendo, divisor, cociente, resto Como Entero

    Escribir "Ingrese el dividendo:"
    Leer dividendo
    Escribir "Ingrese el divisor:"
    Leer divisor

    division(dividendo, divisor, cociente, resto)
    Escribir "Cociente=", cociente, " / Resto=", resto

FinAlgoritmo
```

entonces,

¿diferencia con las funciones?

invocación

- también conocido como *llamada*
- es una transferencia de control
- correspondencia estricta de parámetros respecto de **cantidad**, **orden** y **tipo**

Encuentren los errores suponiendo cad tipo cadena, lg tipo lógico y rl tipo real

```
Funcion cad <- fx( lg, rl)

var_cad = ""
var_cad = fx(3.0,falso)
escribir fx(verdadero,1.5,var_cad)
```

parámetros

- permiten la **comunicación** entre (sub)programas
- hay diferentes formatos, pero la **correspondencia por posición** es más popular que la **correspondencia por nombre**

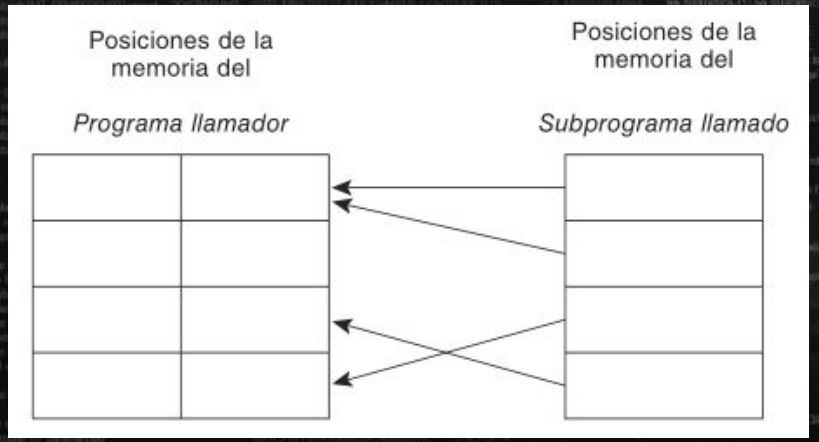
```
SubProceso subpr(un_entero)
...
subpr(34)
...
```

```
SubProceso subpr(nombre,y)
…
//por posición
subpr(“jose”,25)

//por nombre
subpr(y => 25, nombre => "jose")
```

paso de parámetros

- por valor
  - parámetros como variables locales
  - valores del invocador inmutables
  - símil tipo *E*
- por referencia
  - parámetro como referencias de memoria
  - valores del invocador modificables
  - símil tipo *S* o *E/S*

```
A ← 5
B ← 7
llamar_a      PROC1    (A,      18,      B * 3 + 4)
                        5        18       25
procedimiento PROC1 (E entero:   X,   Y,   Z)
```



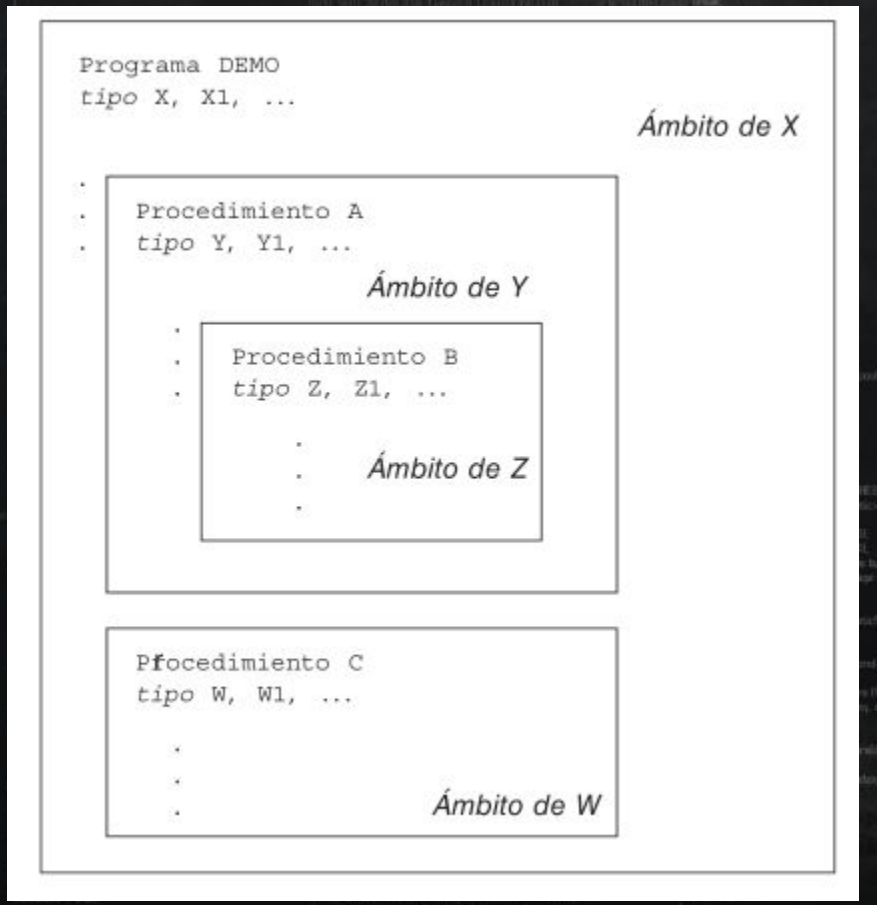
es importante conocer la política de paso de parámetros de un lenguaje

paso de parámetros: ej.

```
//Considere el siguiente código PseInt
//¿qué texto imprime la salida del programa?
Algoritmo PasoParametros
    Definir i, j Como Entero
    i = 2
    j = 3
    sub(i, j)
    Escribir "Principal. i = ", i, " / j = ", j
FinAlgoritmo

SubProceso sub(i Por Valor, j Por Referencia)
    i = i + 10
    j = j + 10
    Escribir "Sub. i = ", i, " / j = ", j
FinSubProceso
```

ámbito

- el **ámbito** determina en **qué partes** del programa una entidad (variable, constante, etc) **puede ser usada**
- **toda variable local es accesible dentro de su ámbito**



ámbito: ejemplo

```
//Considere el siguiente código PseInt
//¿qué texto imprime la salida del programa?
Algoritmo Test_Ambito
    Definir A, B Como Entero
    A = 2
    B = 3
    Escribir "Principal inicio A = ", A, " B = ", B
    sub(A)
    Escribir "Principal fin A = ", A, " B = ", B
FinAlgoritmo

SubProceso sub(B Por Referencia)
    Definir A Como Entero
    A = 10
    Escribir "Sub inicio A = ", A, " B = ", B
    B = A * B
    Escribir "Sub fin A = ", A, " B = ", B
FinSubProceso

///¿Que pasa si defino una variable C en sub y otro subprograma
X quiere modificarlo?
```

variable global

- variable accesible/modificable desde todos los ámbitos
- definición (pseudocódigo):

```
global <tipo_de_dato> <nombre_variable> [=<expresión de inicialización>]
```

- algunos teóricos las consideran nocivas
- pseint no permite su uso

variable global: ej.

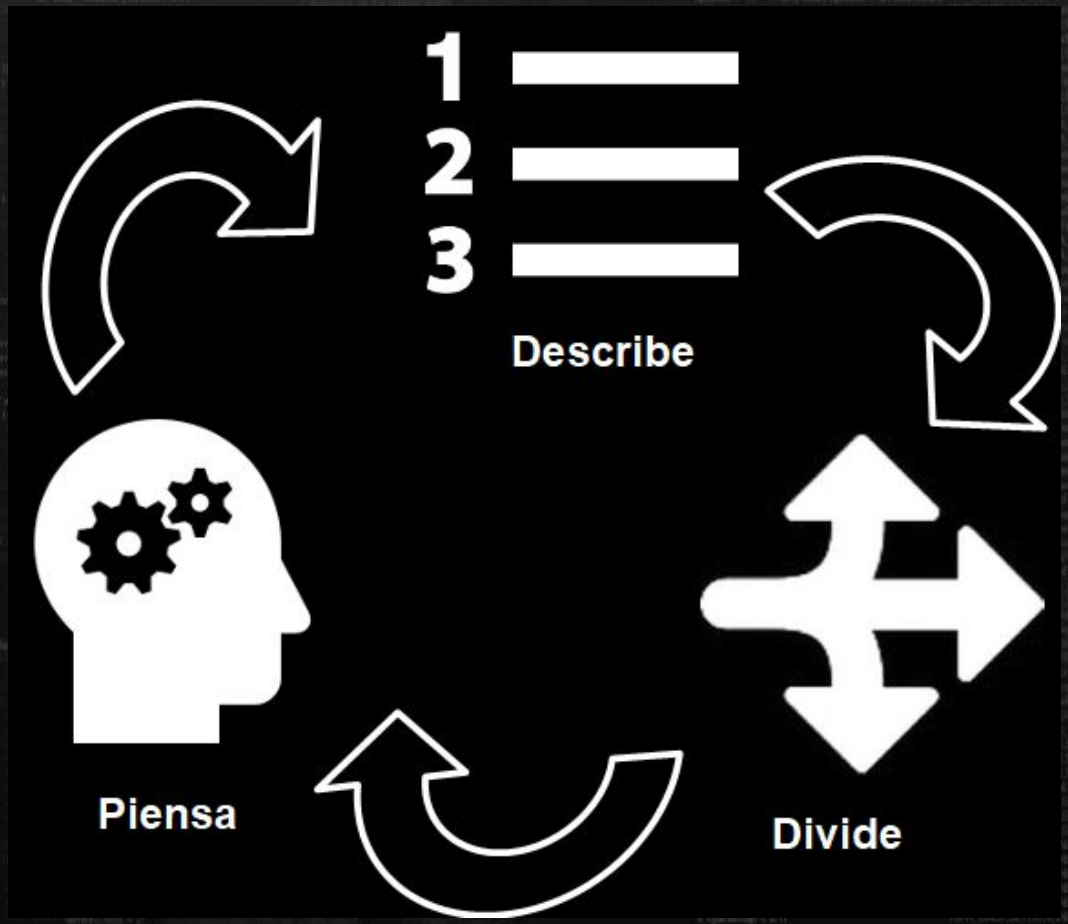
```
//ejemplo de variable global en javascript
let a = 50;

function test(){
    return a + 1;
}

function main()
{
    console.log("Valor a=", a);
    a = test();
    console.log("Nuevo valor a=", a);
}

main();
```

¿a qué ámbito pertenece la variable a?

¿qué valor tiene al finalizar el programa?

diseño algorítmico

descomposición descendente (top-down)



buenas prácticas

- documentar!
- subprogramas con un fin preciso
- nombres de subprogramas y parámetros *autodescriptibles*
- las funciones no deben tener parámetros extra de salida
- evitar variables globales; de no ser posible minimizar los puntos de modificación