Ing. Ventre, Luis O.

CAPITULO 6 (CONT):

FUNCIONES Y VARIABLES

$$y = f(x) = \frac{2x+1}{3}$$

La clase pasada

Ing. Ventre, Luis O.

- Vimos como modularizar nuestro programa con FUNCIONES:

Sin parámetros	Procedimientos
Con 1 parámetro	Que devuelvan 1 valor
Con más de 1 parámetro	Que devuelvan más de 1 valor

- Ahora continuaremos explicando conceptos vinculados al uso de estas funciones….

- 6 Funciones.
  - 6.4 Variables locales y globales.
  - 6.5 El mecanismo de las llamadas funciones.
    - 6.5.1 La pila de las llamadas a función
    - 6.5.2 Paso del resultado de expresiones como arg.
    - 6.5.3 Mas sobre el paso de parámetros
    - 6.5.4 Acceso a variables globales desde funciones

Variables locales y globales

- Las variables que utilizamos en nuestro programa no SON todas iguales.

- Algunas **SON CREADAS** en el programa principal, y otras son CREADAS **adentro** de las funciones.

- Esto clasifica las variables en dos opciones.

GLOBALES

LOCALES

- Veamos esto con un ejemplo:

- Escriba y ejecute el siguiente programa:

```
def funcion():
    s="Hoy es viernes"
    print s


#Programa principal

funcion()
```

```
Hoy es viernes
```

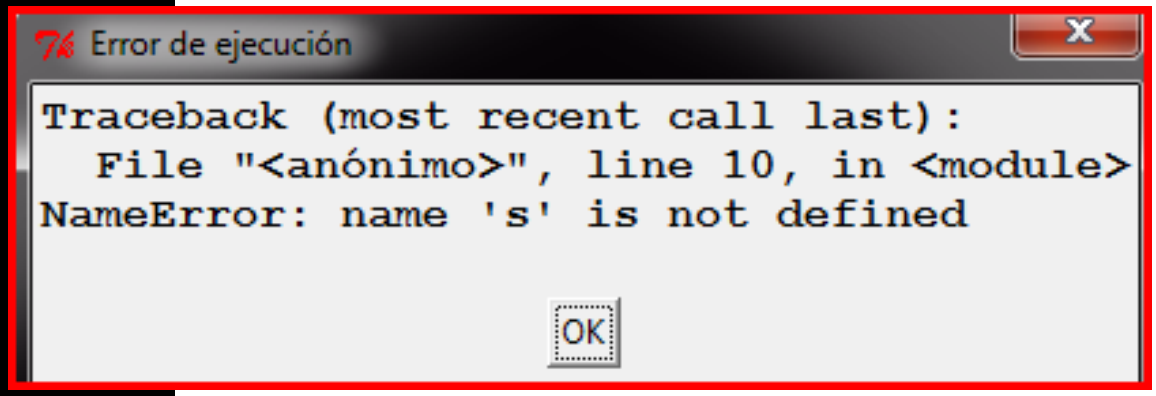
- Agregue a su programa la siguiente línea de código en el programa principal:

```
def funcion():
  s="Hoy es viernes"
  print s


#Programa principal

funcion()

print s
```



- Que sucede?

- Aquí tenemos un primer concepto. Las variables creadas adentro de una FUNCION son **LOCALES**

- **NO PUEDEN ACCEDERSE** desde el programa principal

- Estas variables se **crean al ejecutar** la función y se **destruyen** al **salir** de la misma!!!

```
def funcion():
  s="Hoy es viernes"
  print s


#Programa principal

funcion()

print s
```

- Las variables que solo existen en el cuerpo de una función se denominan **VARIABLES LOCALES**.

- El resto de las variables se llaman **GLOBALES**.

- Los **parámetros** de una función

TAMBIEN SON VARIABLES LOCALES

```
def funcion(a,b):
  a=0
  b=1
  s="Hoy es viernes"
  print s


#Programa principal

funcion(10,20)

print  a
```

- Veamos el siguiente ejercicio:

- Escriba y ejecute el siguiente programa:

```
def funcion():
    #s="Adentro de Funcion"
    print s


#Programa principal
s="Estamos en el programa principal"

funcion()
```

- Ahora **DES-comente** la primer línea de la función y ejecute, que sucede? Porque?.

```
def funcion():
  s="Adentro de Funcion"
  print s


#Programa principal
s="Estamos en el programa principal"

funcion()
```

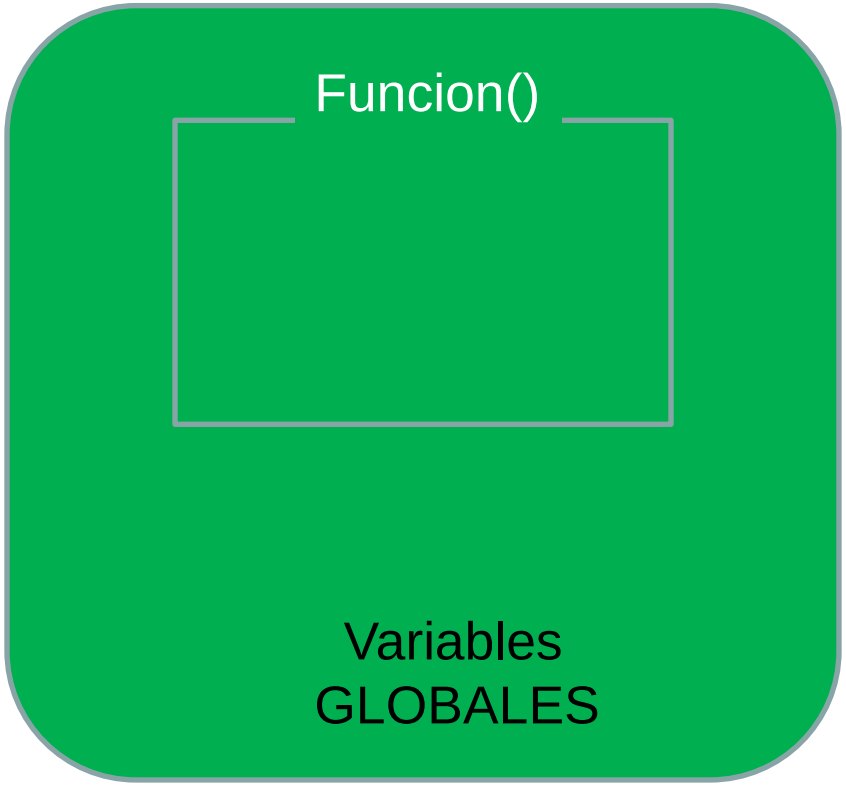
- En el primer caso, desde la función se imprimió una **variable GLOBAL**. Ya que es la única variable con ese nombre.

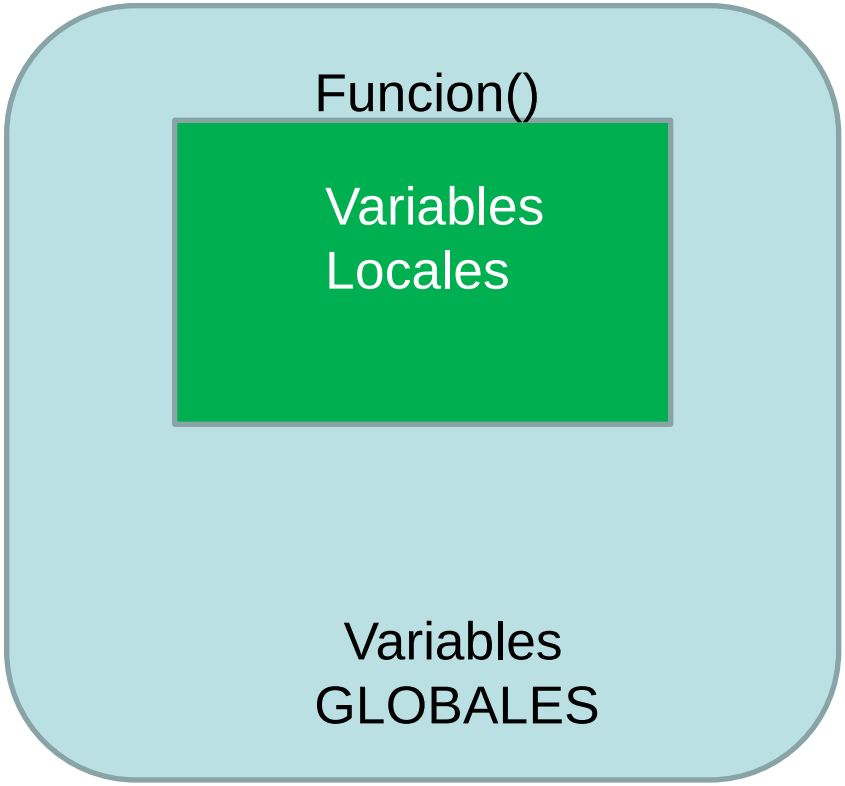
- En el segundo caso, existen **DOS** variables con el **MISMO NOMBRE.**

- Python, identifica una variable como LOCAL y la otra como GLOBAL.

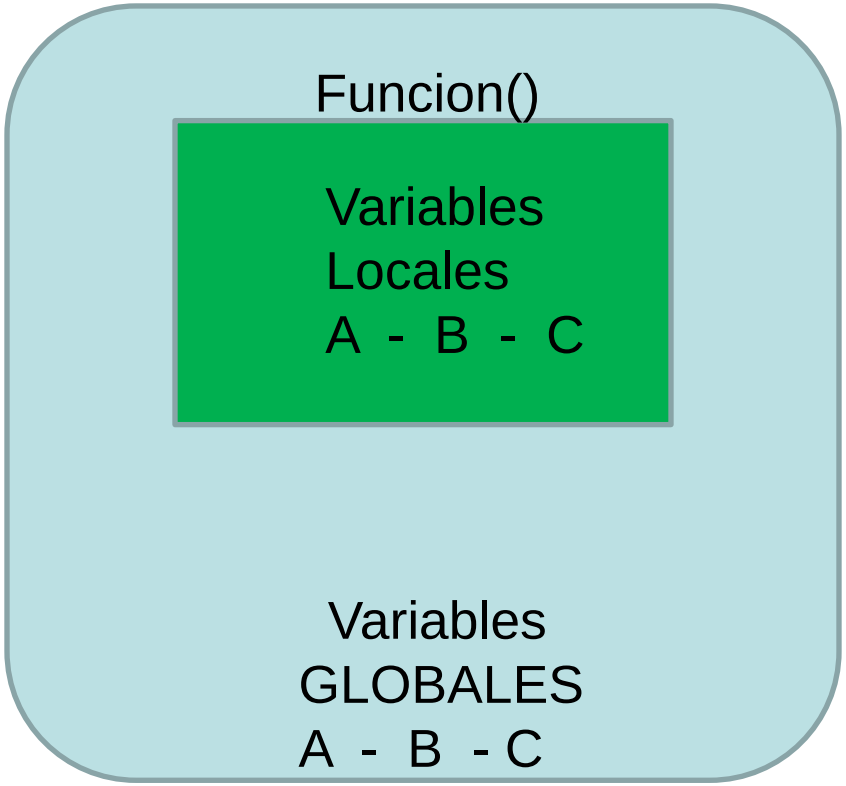
- Siempre que se acceda desde la función, hará uso de la VARIABLE LOCAL. A menos que se indique explícitamente lo contrario…

- Podemos hablar de regiones VISIBLES de las variables





- Pero si existen variables de igual nombre cada una existe en su espacio propio.



- Veamos que hicimos la clase pasada sin tanto conocimiento:

```
lpares=lista_pares(lista)
```

Llamado a función

```
def lista_pares(lista):
 lpares=[]
 for i in lista:
     if i%2==0:
         lpares.append(i)
 return lpares
```

Recibe en lista, parámetro local!.

Crea otra lista LOCAL.

Devuelve la lista para que se almacene en prog. ppal

- 6 Funciones.
  - 6.4 Variables locales y globales.
  - 6.5 El mecanismo de las llamadas funciones.
    - 6.5.1 La pila de las llamadas a función
    - 6.5.2 Paso del resultado de expresiones como arg.
    - 6.5.3 Mas sobre el paso de parámetros
    - 6.5.4 Acceso a variables globales desde funciones

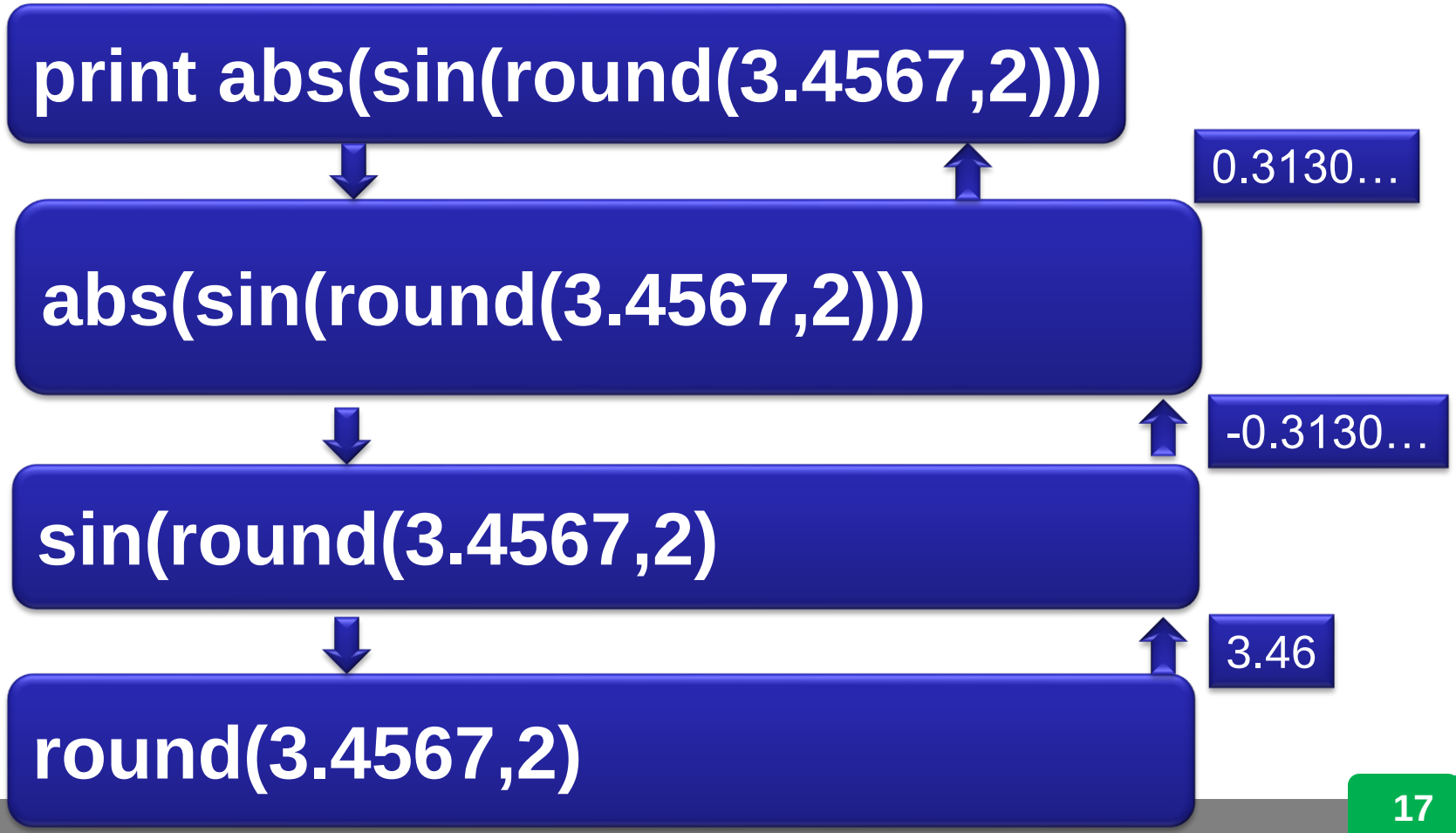
- Hemos observado, que desde una FUNCION podemos invocar a otra FUNCION. Desde esta ultima, podemos invocar a OTRA FUNCION.

- Cada vez, que se produce una nueva llamada, la ejecución anterior queda detenida a la espera de un resultado.

```
print  abs(sin(round(3.4567,2)))
```

- **Como seria la secuencia de ejecución y el ARBOL de llamadas de las funciones:**

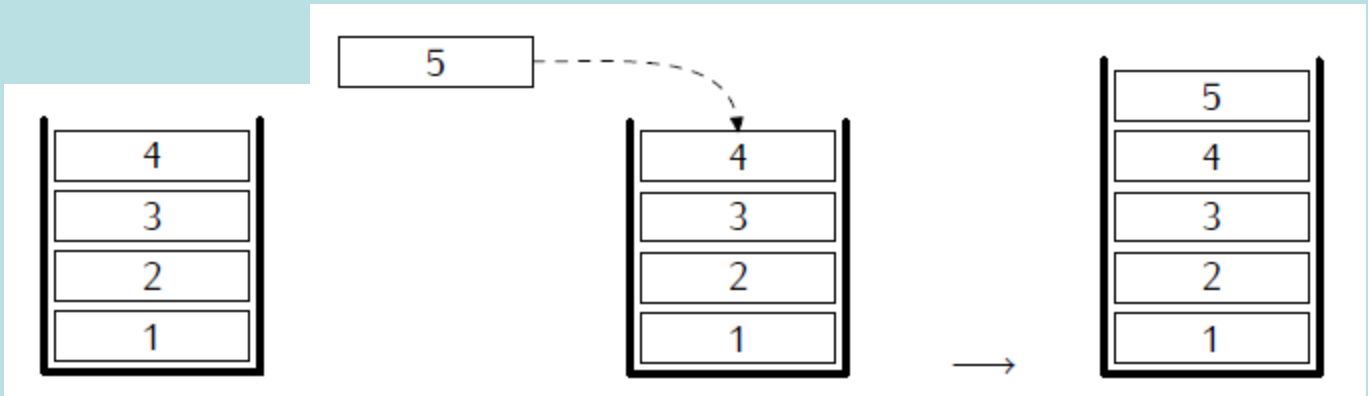
- La ejecución comenzaría en el programa principal



- 6 Funciones.
  - 6.4 Variables locales y globales.
  - 6.5 El mecanismo de las llamadas funciones.
    - 6.5.1 La pila de las llamadas a función
    - 6.5.2 Paso del resultado de expresiones como arg.
    - 6.5.3 Mas sobre el paso de parámetros
    - 6.5.4 Acceso a variables globales desde funciones

- **Python utiliza internamente una estructura especial de memoria para recordar los datos al llamar a las funciones.**

- **Esta estructura es del tipo PILA. Donde se pueden poner y sacar elementos pero solo por su parte superior.**



- **De esta forma, en cada llamado a función se agrega un bloque nuevo.**

- **Este bloque contiene toda la información necesaria para regresar cuando finalice la ejecución de la función**

- **Y además el espacio de memoria correspondiente para las variables LOCALES de la misma.**

Veamos un ejemplo concreto:

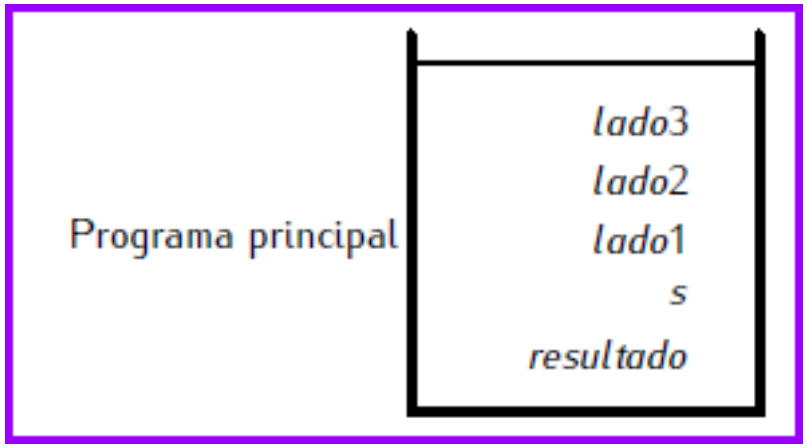
- **Tenemos el siguiente programa de calculo de área**

```
1  from math import sqrt, asin, pi
2
3  def area_triangulo(a, b, c):
4      s = (a + b + c) / 2.0
5      return sqrt(s * (s-a) * (s-b) * (s-c))
6
7  def angulo_alfa(a, b, c):
8      s = area_triangulo(a, b, c)
9      return 180 / pi * asin(2.0 * s / (b*c))
10
11 def menu():
12     opcion = 0
13     while opcion != 1 and opcion != 2:
14         print '1) Calcular área del triángulo'
15         print '2) Calcular ángulo opuesto al primer lado'
16         opcion = int(raw_input('Escoge opción: '))
17     return opcion
18
19 lado1 = float(raw_input('Dame lado a: '))
20 lado2 = float(raw_input('Dame lado b: '))
21 lado3 = float(raw_input('Dame lado c: '))
22
23 s = menu()
24
25 if s == 1:
26     resultado = area_triangulo(lado1, lado2, lado3)
27 else:
28     resultado = angulo_alfa(lado1, lado2, lado3)
29
30 print 'Escogiste la opción', s
31 print 'El resultado es:', resultado
```

- **Veamos paso a paso una ejecución posible:**

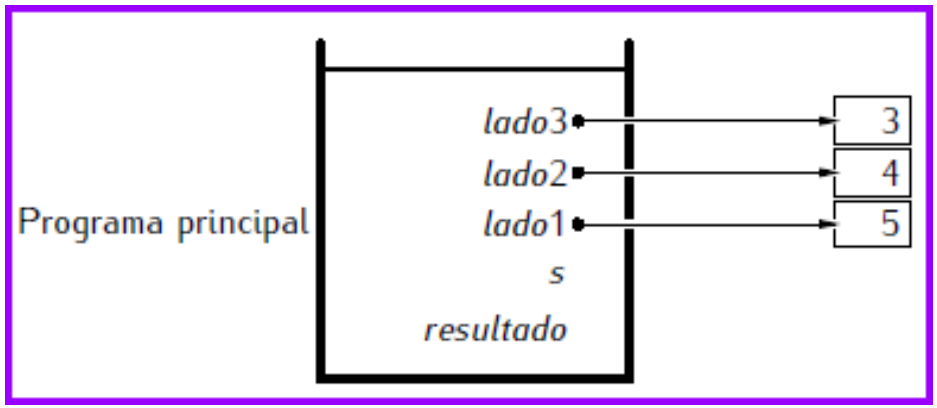
```
Dame lado a: 5
Dame lado b: 4
Dame lado c: 3
1) Calcular área del triángulo
2) Calcular ángulo opuesto al primer lado
Escoge opción: 2
Escogiste la opción 2
El resultado es: 90.0
```

- **Cuando el programa arranca Python nos reserva el siguiente espacio en la pila:**

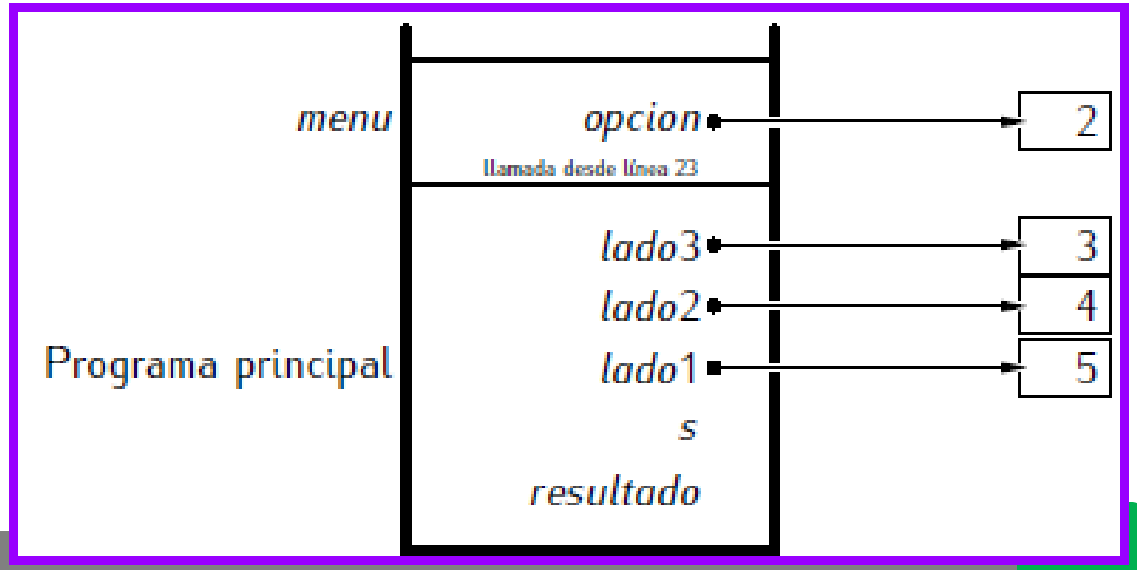


Ing. Ventre, Luis O.

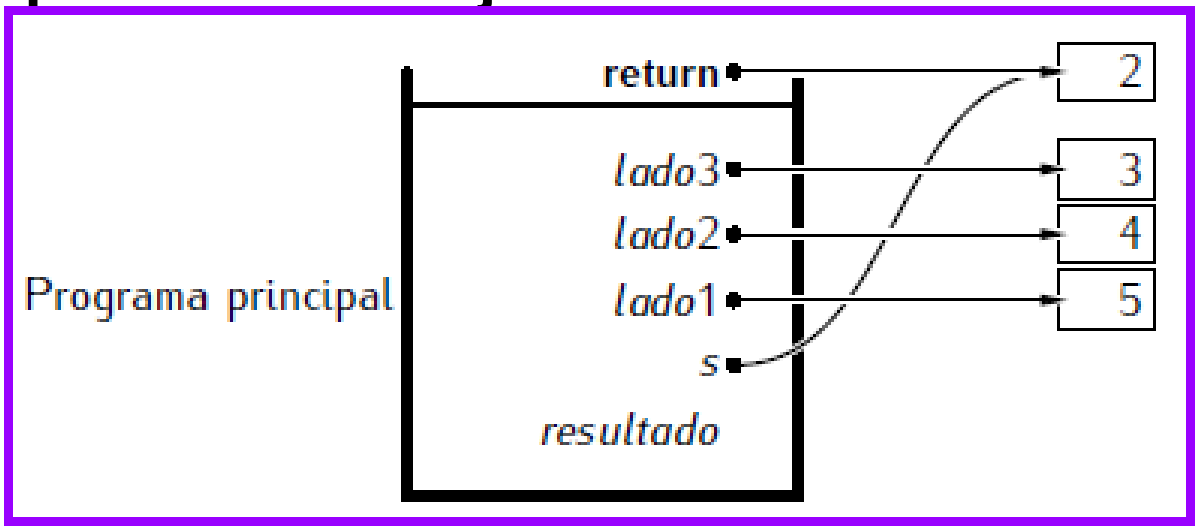
- **El usuario ingresa datos y la memoria queda:**



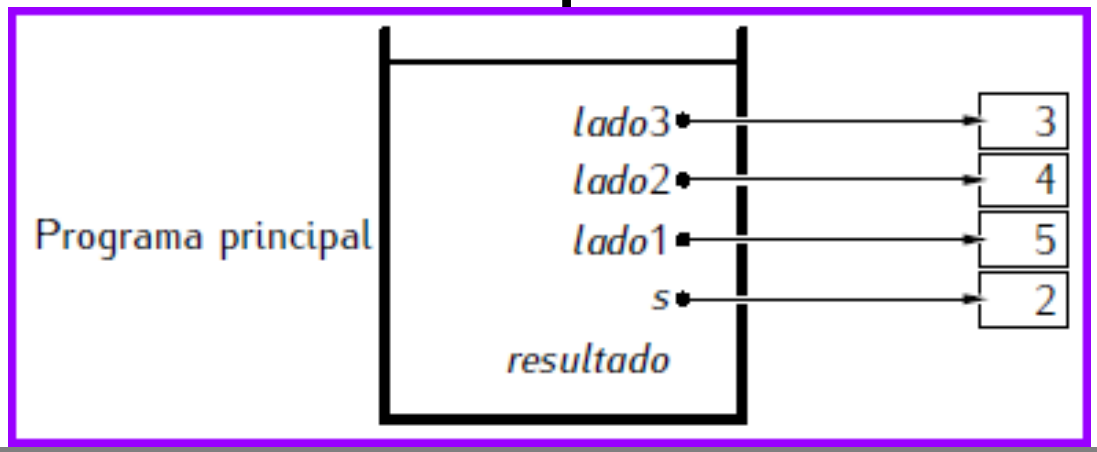
- **Entonces se produce el llamado a función MENU.**
- **Se reserva un espacio nuevo:**



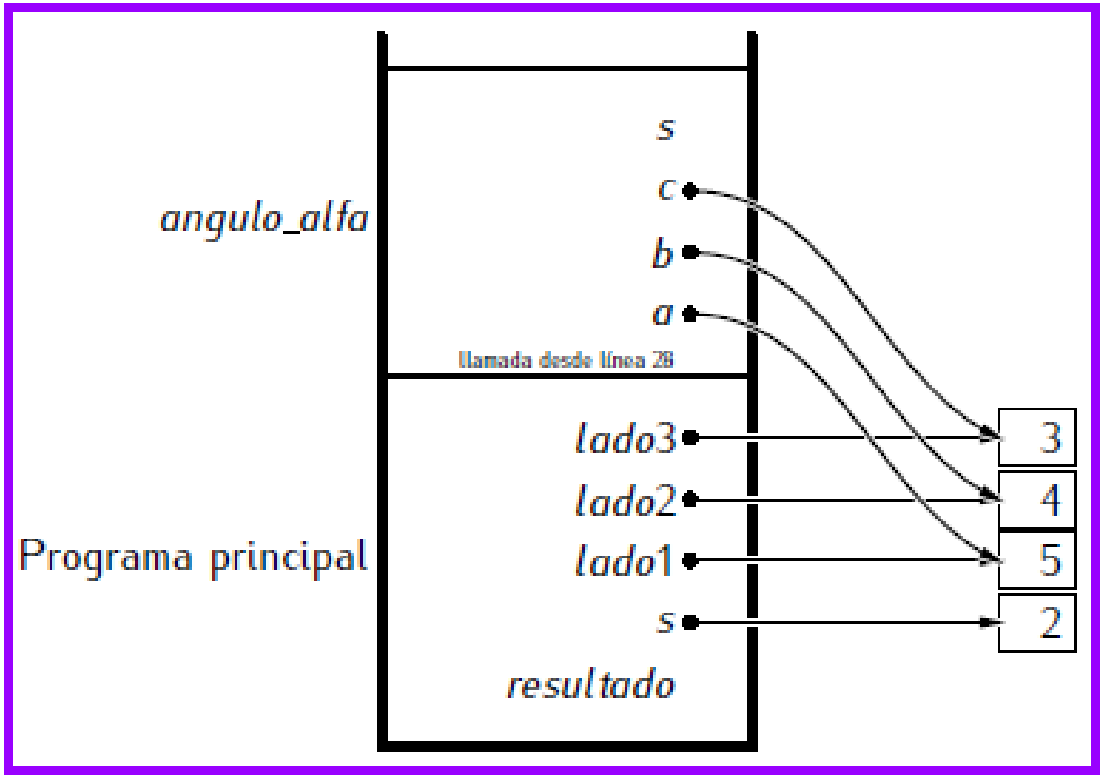
- **Una vez que finalizo la ejecución de MENU se elimina:**



- **Queda la referencia al valor que devuelve la función.**

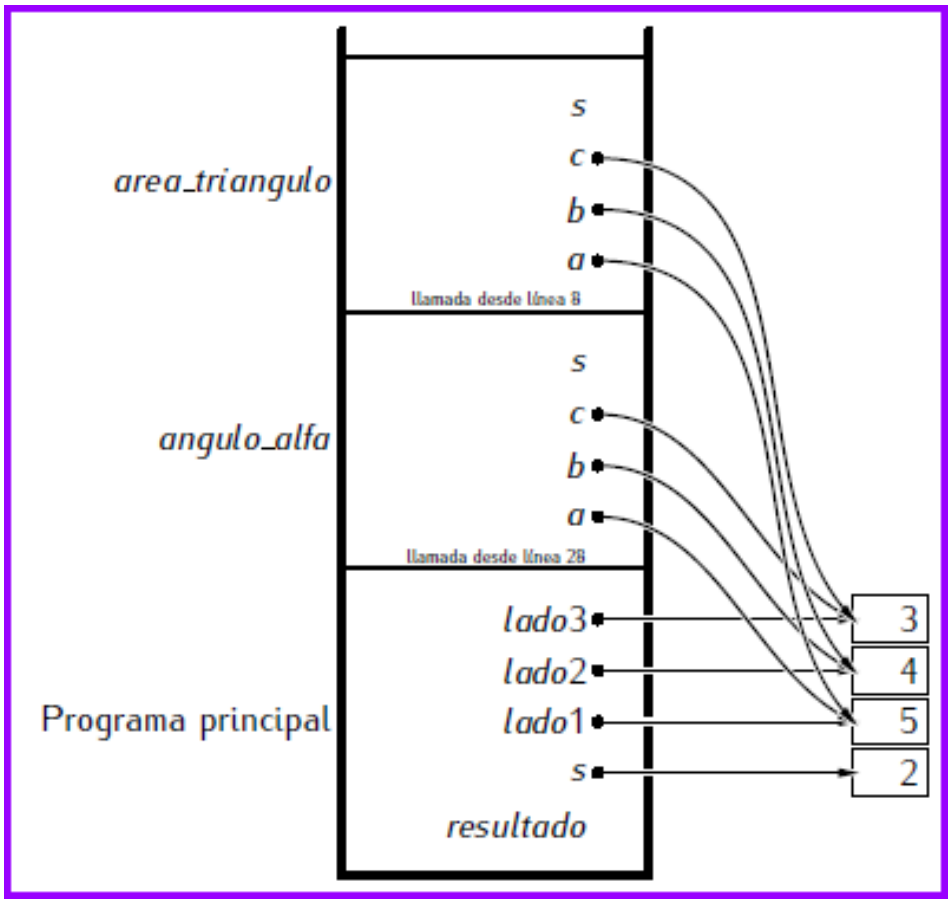


- **Continua la ejecución y se llama a ANGULO_ALFA:**



- **En el paso de parámetros a b y c apuntan a los valores de lado1 lado 2 y lado 3.**

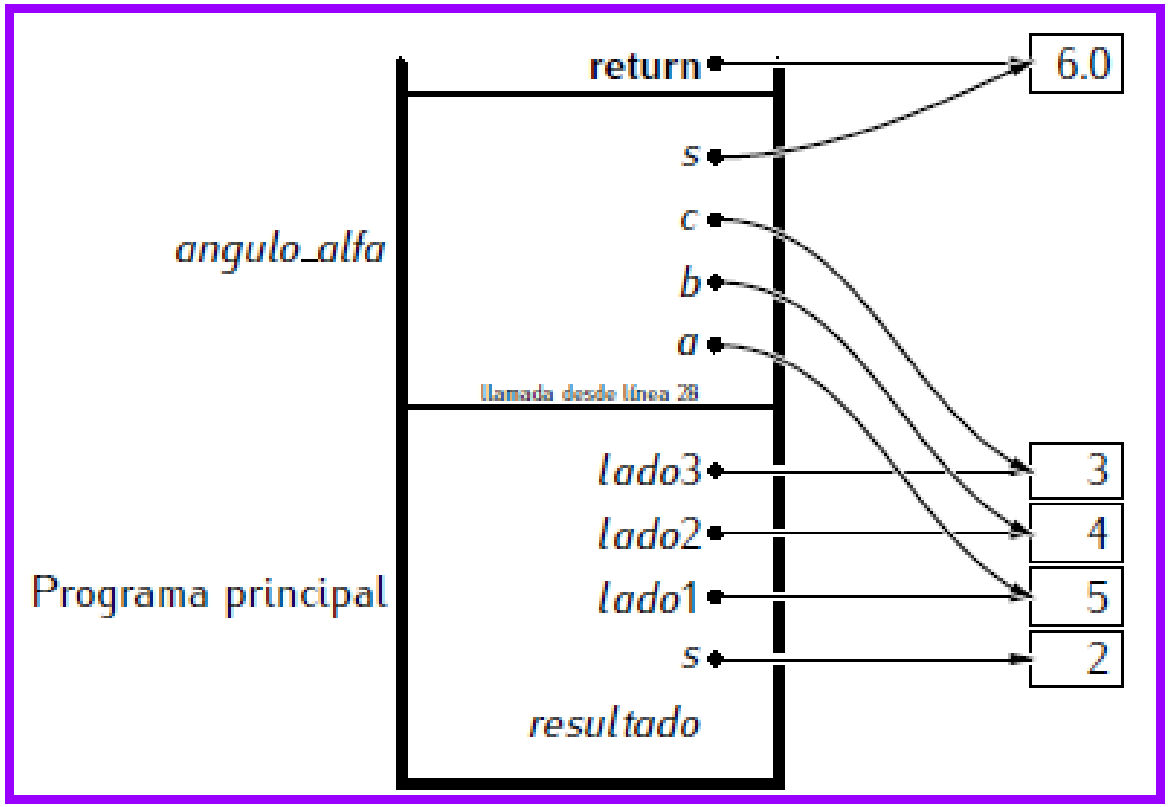
- **Desde ANGULO_ALFA se invoca a AREA_TRIANGULO:**



- **Mismos identificadores en las funciones, pero espacios de memoria diferentes.**

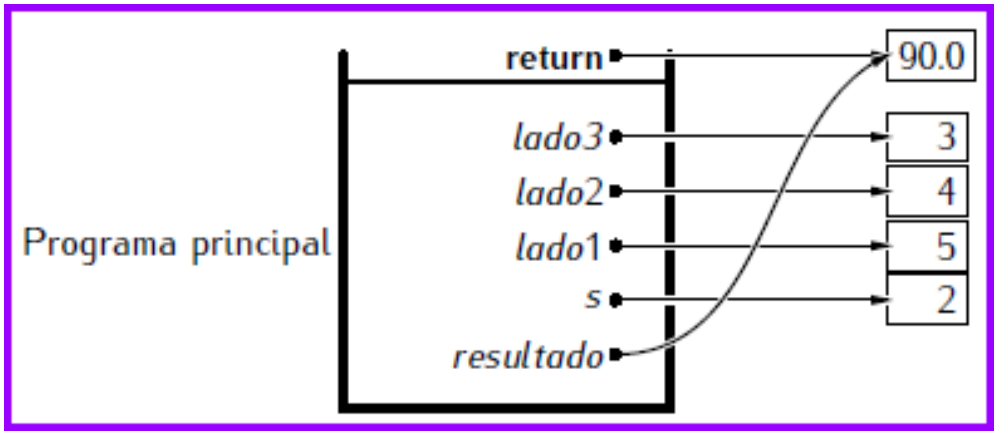
Ing. Ventre, Luis O.

- **Finaliza AREA_TRIANGULO devolviendo el resultado:**

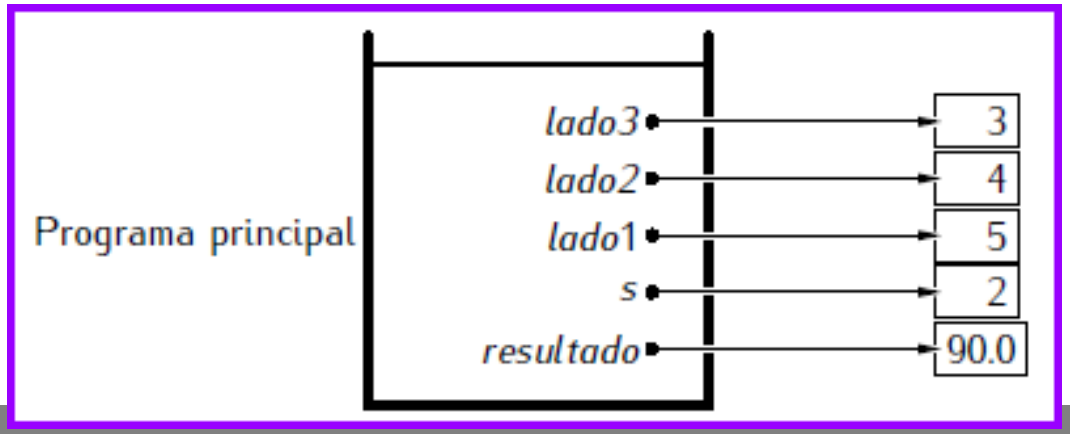


- **Luego continuara el calculo de ángulo alfa. Y posteriormente devolverá el resultado a prog. ppal.**

- **ANGULO_ALFA devuelve el resultado**



- **Finaliza el programa. Observa que la variable global S, siempre valió 2!!**



- 6 Funciones.
  - 6.4 Variables locales y globales.
  - 6.5 El mecanismo de las llamadas funciones.
    - 6.5.1 La pila de las llamadas a función
    - 6.5.2 Paso del resultado de expresiones como argumento.
    - 6.5.3 Mas sobre el paso de parámetros
    - 6.5.4 Acceso a variables globales desde funciones

Resultado de expresiones como argumento

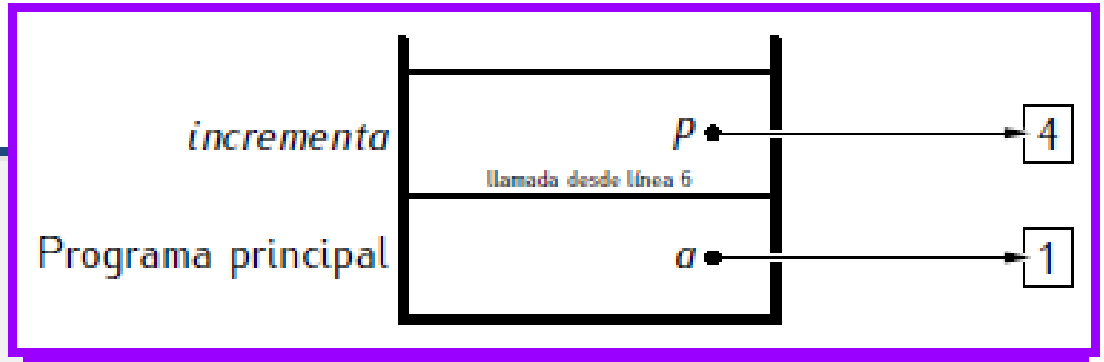
- Observamos como era el funcionamiento de la PILA cuando enviamos variables en el argumento de las funciones. Si enviamos un resultado de expresión como será?

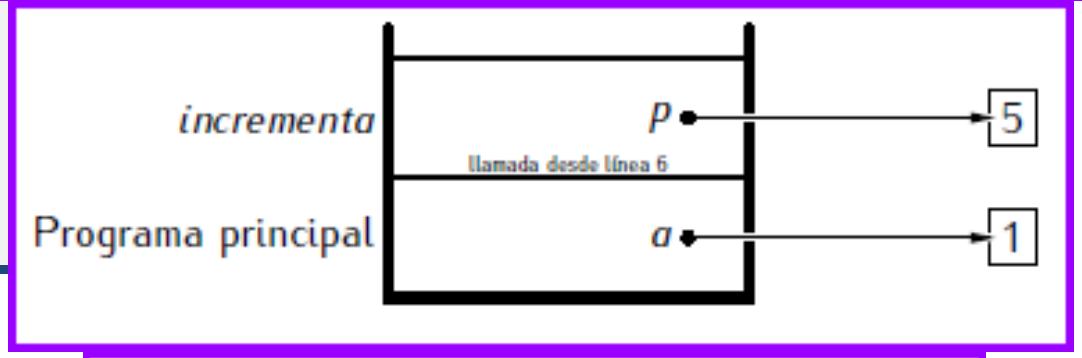
```
1 def incrementa(p):
2     p = p + 1
3     return p
4
5 a = 1
6 a = incrementa(2+2)
7 print 'a:', a
```

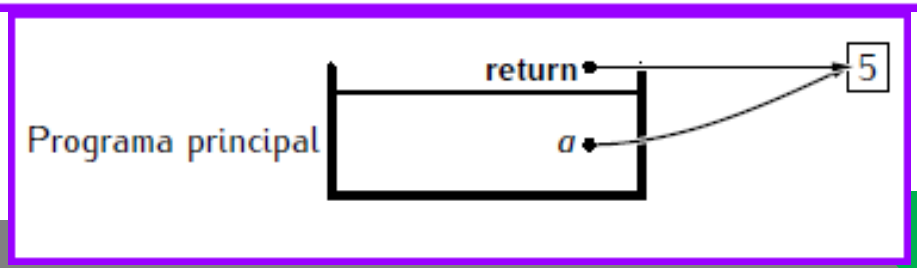
Resultado de expresiones como argumento

- El parámetro p, apunta a un nuevo espacio de memoria que almacena el RESULTADO DE LA EXPRESION.

```
1 def incrementa(p):
2     p = p + 1
3     return p
4
5 a = 1
6 a = incrementa(2+2)
7 print 'a:', a
```







- 6 Funciones.
  - 6.4 Variables locales y globales.
  - 6.5 El mecanismo de las llamadas funciones.
    - 6.5.1 La pila de las llamadas a función
    - 6.5.2 Paso del resultado de expresiones como argumento.
    - 6.5.3 Mas sobre el paso de parámetros
    - 6.5.4 Acceso a variables globales desde funciones

- Hemos visto que en el paso de parámetros, cada parámetro **apunta a un LUGAR DE MEMORIA**.
- Este lugar puede estar **ya apuntado por OTRA VARIABLE**.
- Que ocurre si el **parámetro es modificado** DENTRO DE LA FUNCION?...varia el argumento también?
- La respuesta varia depende el caso….
- Veamos el primero con un ejemplo simple:

- Copie y ejecute el siguiente código:

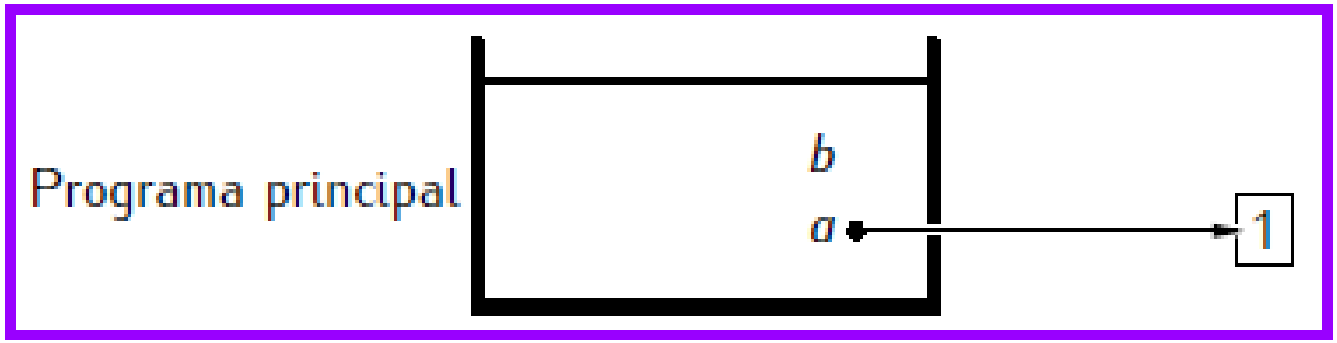
```
1 def incrementa(p):
2     p = p + 1
3     return p
4
5 a = 1
6 b = incrementa(a)
7
8 print 'a:', a
9 print 'b:', b
```

- Antes de ejecutarlo intente resolverlo a mano, y predecir el resultado

- La ejecución es:

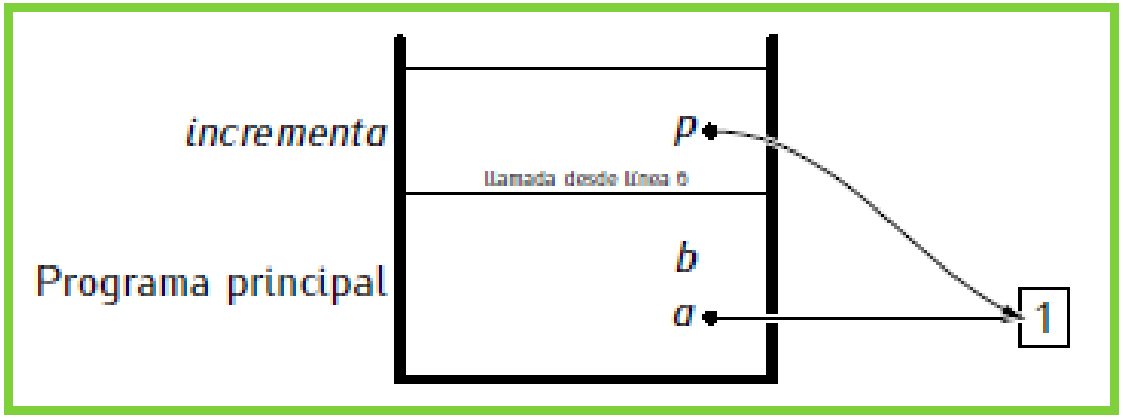
```
a:  1
b:  2
```

- Pero no debería modificarse el parámetro? Veamos la PILA

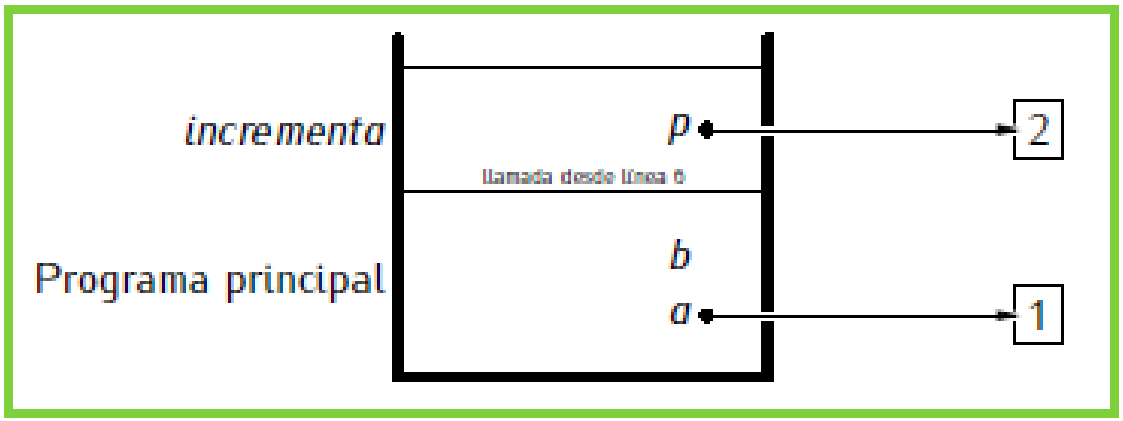


- Cuando invocamos INCREMENTA, el parámetro p recibe una referencia al valor apuntado por a

- La pila queda:



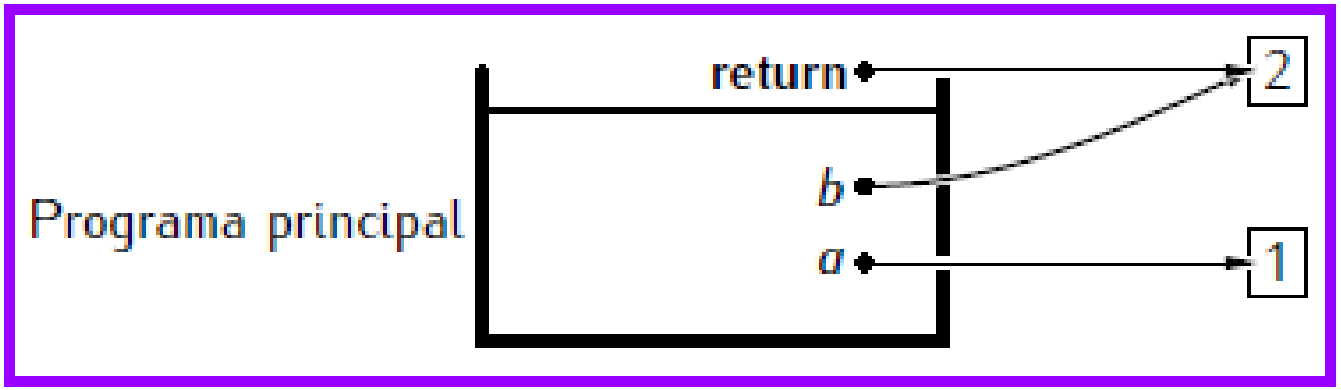
- En la función se asigna un nuevo valor a p…quedando:



- Recuerda como funcionaba la asignación en python.

  - Primero se evalúa el valor a la derecha del signo «=»
  - Segundo se hace que la parte izquierda del signo, apunte al resultado.

- De esta forma luego de la ejecución b, recibe una referencia al numero 2.



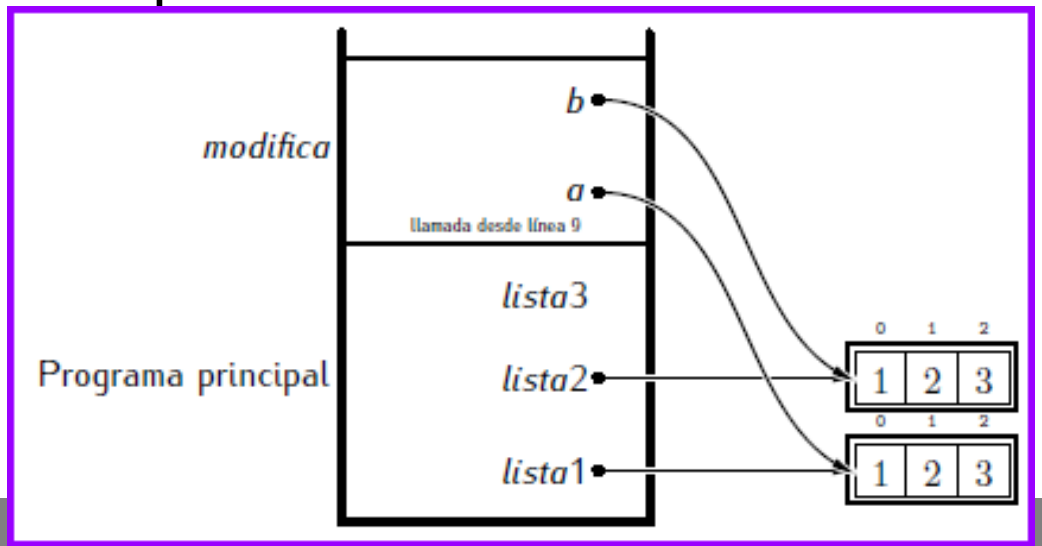
- Veamos que sucederá en el siguiente caso:

```
1  def modifica(a, b):
2      a.append(4)
3      b = b + [4]
4      return b
5
6  lista1 = [1, 2, 3]
7  lista2 = [1, 2, 3]
8
9  lista3 = modifica(lista1, lista2)
10
11 print lista1
12 print lista2
13 print lista3
```

- El resultado de la ejecución es:

```
[1, 2, 3, 4]
[1, 2, 3]
[1, 2, 3, 4]
```

- Esto implica que el primer argumento se HA MODIFICADO?.
- Veamos paso a paso:



- Luego del llamado en la función se modifica:

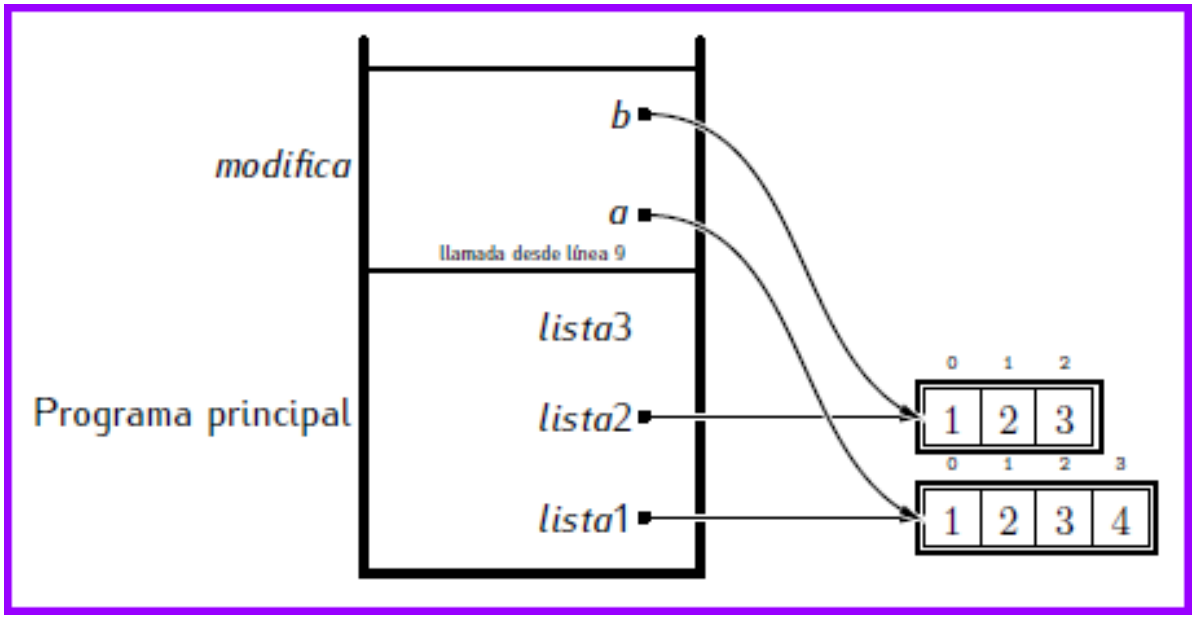
- Línea 2, tiene una instrucción **APPEND**, la cual hace **CRECER a la lista APUNTADA** por el final.

- Como la lista modificada es apuntada por el parámetro a, y por la variable global lista1 **ambas se ven afectadas**.

- En la Línea 3, hay una **asignación con concatenación** de un elemento, esto hace reservar un nuevo espacio de memoria para cuatro elementos.

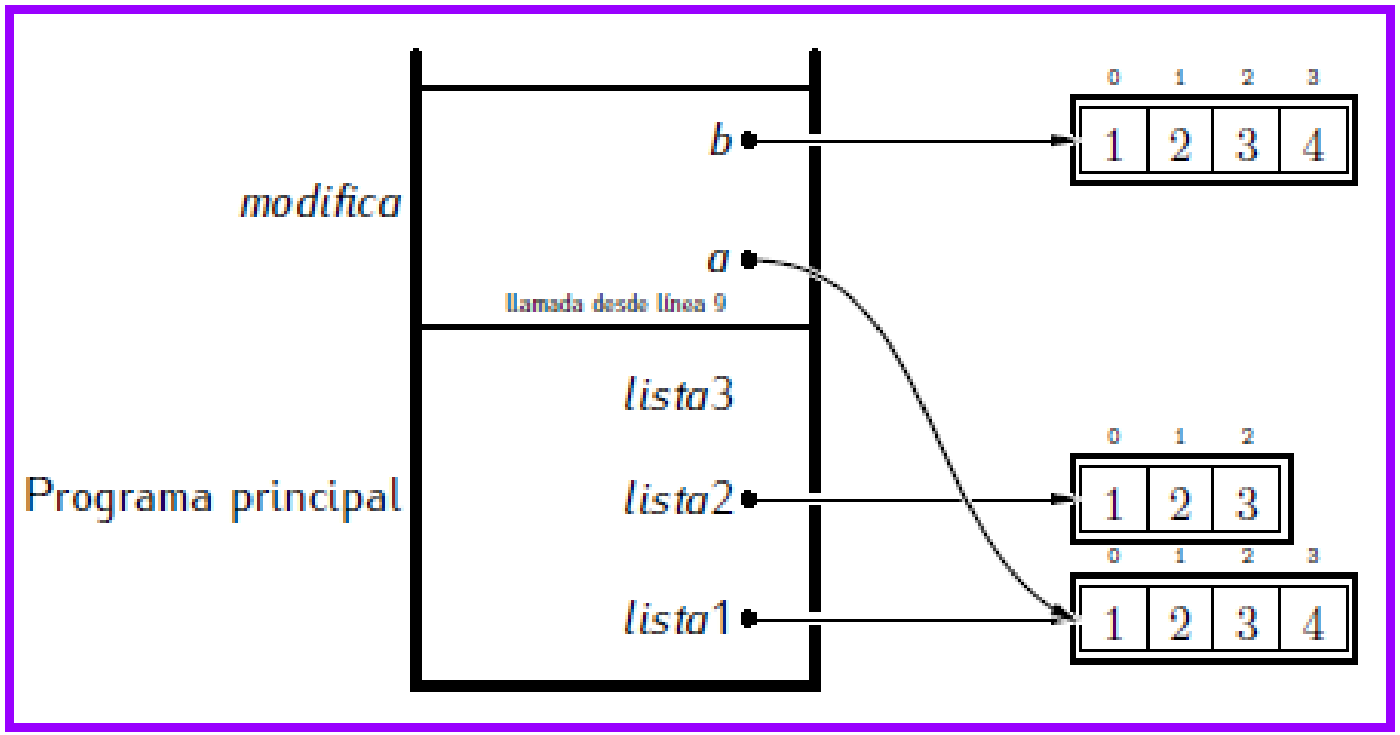
- Una vez creada la nueva lista, se hace que se apunte.

- Veamos como se modifica la PILA paso a paso:



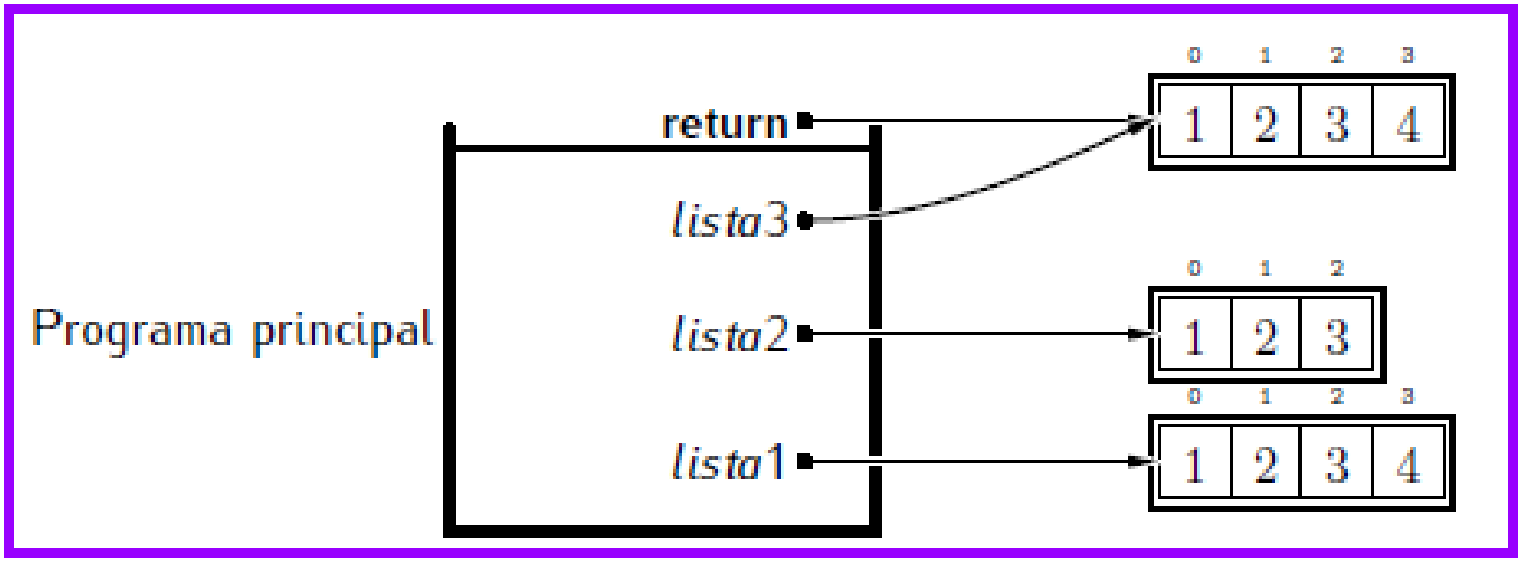
- Estado luego de la ejecución del APPEND.

- Veamos como se modifica la PILA paso a paso:



- Estado luego de la ejecución de la asignación.

- RESULTADO FINAL.



- Recuerda:
  - *Si un parámetro modifica su valor con una asignación, obtendrá una nueva zona de memoria y perderá relación el argumento que le dio origen.*
  - *Operaciones como* **append, del o asignaciones a elementos indexados**, *modifican la propia lista, por lo que se afecta el parámetro y el argumento!.*
  - *Las cadenas son inmutables, no pueden cambiar su valor mediante operaciones como DEL, APPEND, o asignacion directa a un elemento indexado.*

- Un ultimo caso, veamos un procedimiento que invierta una lista. Observa que no devuelve NADA, modifica la lista original

```
1 def invierte(lista):
2     for i in range(len(lista)/2):
3         c = lista[i]
4         lista[i] = lista[len(lista)-1-i]
5         lista[len(lista)-1-i] = c
6
7 a = [1, 2, 3, 4]
8 invierte(a)
9 print a
```

- *En el procedimiento hay asignaciones a objetos indexados, lo que modifica la lista original. PORQUE len(lista)/2??*

- 6 Funciones.
  - 6.4 Variables locales y globales.
  - 6.5 El mecanismo de las llamadas funciones.
    - 6.5.1 La pila de las llamadas a función
    - 6.5.2 Paso del resultado de expresiones como argumento.
    - 6.5.3 Mas sobre el paso de parámetros
    - 6.5.4 Acceso a variables globales desde funciones

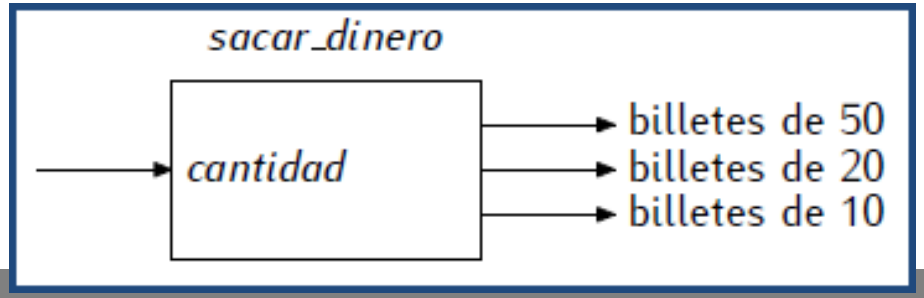
Acceso a variables globales desde funciones

Ing. Ventre, Luis O.

- Luego de todo lo visto, podríamos creer que en las funciones solo pueden usarse VARIABLES LOCALES?...y esto no es así

- En python podemos desde funciones usar y modificar **variables globales**, SOLO HAY QUE INDICARLE al interprete que lo SON.

- Veremos como con un ejemplo, de un programa que implemente una función de cajero automático. El cual devuelva la cantidad de billetes a entregar de acuerdo a lo solicitado



Acceso a variables globales desde funciones

- Veamos el código

```
1  carga50 = 100
2  carga20 = 100
3  carga10 = 100
4
5  def sacar_dinero(cantidad):
6      de50 = cantidad / 50
7      cantidad = cantidad % 50
8      de20 = cantidad / 20
9      cantidad = cantidad % 20
10     de10 = cantidad / 10
11     carga50 = carga50 - de50
12     carga20 = carga20 - de20
13     carga10 = carga10 - de10
14     return [de50, de20, de10]
```

Que ocurrirá con las variables

carga50
carga20
carga10…?

Acceso a variables globales desde funciones

- Si ejecutamos esta función obtendremos:

```
1  carga50 = 100
2  carga20 = 100
3  carga10 = 100
4
5  def
6      d
7      c
8      d
9      c
10     d
11     carga50 = carga50 - de50
12     carga20 = carga20 - de20
13     carga10 = carga10 - de10
14     return [de50, de20, de10]
```

```
$ python cajero.py↵
Cantidad a extraer: 70
Traceback (most recent call last):
  File "cajero.py", line 17, in ?
    print sacar_dinero(c)
  File "cajero.py", line 11, in sacar_dinero
    carga50 = carga50 - de50
UnboundLocalError: local variable 'carga50' referenced before assignment
```

Acceso a variables globales desde funciones

- Python usa una regla simple:
  - Si adentro de la función se le asigna un valor es LOCAL
  - Sino es GLOBAL.

- Si una variable es local y no fue definido su valor antes tenemos el error visto.

- Debemos indicarle a python que USE las variables GLOBALES

```
3 carga10 = 100
4
5 def sacar_dinero(cantidad):
6     global carga50, carga20, carga10
7     de50 = cantidad / 50
```

Acceso a variables gl

- Ahora si,

la versión completa del código será:

```
1  carga50 = 100
2  carga20 = 100
3  carga10 = 100
4
5  def sacar_dinero(cantidad):
6      global carga50, carga20, carga10
7      if cantidad <= 50 * carga50 + 20 * carga20 + 10 * carga10:
8          de50 = cantidad / 50
9          cantidad = cantidad % 50
10         if de50 >= carga50: # Si no hay suficientes billetes de 50
11             cantidad = cantidad + (de50 - carga50) * 50
12             de50 = carga50
13         de20 = cantidad / 20
14         cantidad = cantidad % 20
15         if de20 >= carga20: # y no hay suficientes billetes de 20
16             cantidad = cantidad + (de20 - carga20) * 20
17             de20 = carga20
18         de10 = cantidad / 10
19         cantidad = cantidad % 10
20         if de10 >= carga10: # y no hay suficientes billetes de 10
21             cantidad = cantidad + (de10 - carga10) * 10
22             de10 = carga10
23         # Si todo ha ido bien, la cantidad que resta por entregar es nula:
24         if cantidad == 0:
25             # Así que hacemos efectiva la extracción
26             carga50 = carga50 - de50
27             carga20 = carga20 - de20
```

Repasando!...

- **Variables Locales y Variables Globales.**
- **Donde pueden usarse o son visibles las variables locales y donde las globales.**
- **El mecanismo de las llamadas a función.**
- **La PILA de las llamadas y el paso de parámetros.**

- **Paso del resultado de expresiones como argumentos. Como se comporta la pila cuando la referencia es a un numero y no a una variable.**

- **Mas sobre el paso de parámetros…se modifica también el argumento o no?**
  - **Con asignaciones?**
  - **Con append y subindices?**

- **Acceso a variables globales desde FUNCIONES.**

Ejercicio final

- **Realice un programa, que:**

1. A través de un PROCEDIMIENTO llamado **cargar_datos(lista) que reciba una lista,** permita al usuario ingresar una lista de 4 elementos. Esta función NO debe DEVOLVER la lista al programa principal. **SE DEBE MODIFICAR la lista declarada en main con append**.

2. Desde el programa principal deberá imprimir la lista luego de ingresada.

Ejercicio final

3. Su programa deberá implementar una función llamada **lista_nonulos(lista)**, que reciba la lista original de 4 elementos y genere una nueva lista LOCAL con los elementos que sean diferentes de 0. Esta lista debe ser devuelta al programa principal.

4. Desde el **programa principal** deberá imprimir la lista de elementos no nulos recibida de la función del punto 3.