Ing. Ventre, Luis O.

CAPITULO 6:

FUNCIONES

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

- En los capítulos anteriores hemos aprendido a usar funciones predefinidas:

abs( )

round( )

sin( )

- Ahora veremos como crear **NUESTRAS** propias funciones. Para enriquecer y adaptar el lenguaje a nuestras necesidades y resolver problemas complejos de manera simple.

- Llamando a funciones desde nuestro programa.

- 6 Funciones.
  - 6.1 Uso de funciones
  - 6.2 Definición de funciones
    - 6.2.1 Funciones con un solo parámetro
    - 6.2.2 Funciones con varios parámetros
    - 6.2.3 Funciones sin parámetros
    - 6.2.4 Procedimientos: Funciones sin devolución.
    - 6.2.5 Funciones que devuelven LISTAS.

- Denominamos: **ACTIVAR, LLAMAR o INVOCAR** a una función, a la acción de utilizarla.

- Las funciones que hemos visto, reciben cero, o mas datos (argumentos) para trabajar, separados con comas y entre «( )».

- Estas puede , veamos ejemplos:

```
>>> abs(-3) ↲
3
>>> abs(round(2.45, 1)) ↲
2.5
>>> from sys import exit ↲
>>> exit() ↲
```

- También podíamos llamar a una función desde una expresión, teniendo en cuenta los tipos de datos!

```
>>> 2.5 / abs(round(2.45, 1))
1.0
>>> 3 + str(3)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: number coercion failed
```

- Ahora veremos como definir nuestras propias funciones!.

- 6 Funciones.
  - 6.1 Uso de funciones
  - 6.2 Definición de funciones
    - 6.2.1 Funciones con un solo parámetro
    - 6.2.2 Funciones con varios parámetros
    - 6.2.3 Funciones sin parámetros
    - 6.2.4 Procedimientos: Funciones sin devolución.
    - 6.2.5 Funciones que devuelven LISTAS.

- 6 Funciones.
  - 6.1 Uso de funciones
  - 6.2 Definición de funciones
    - 6.2.1 Funciones con un solo parámetro
    - 6.2.2 Funciones con varios parámetros
    - 6.2.3 Funciones sin parámetros
    - 6.2.4 Procedimientos: Funciones sin devolución.
    - 6.2.5 Funciones que devuelven LISTAS.

- Funciones con un solo parámetro:

- Veamos COMO **definir una función** sencilla, que reciba un valor y **devuelva** el resultado de elevarlo al cuadrado.

```
def cuadrado(x):
    return x ** 2
```

- **Palabras nuevas!: DEF significa DEFINIR.**
  RETURN significa DEVOLVER.

- **Una vez DEFINIDA** ya PODEMOS utilizar la función!

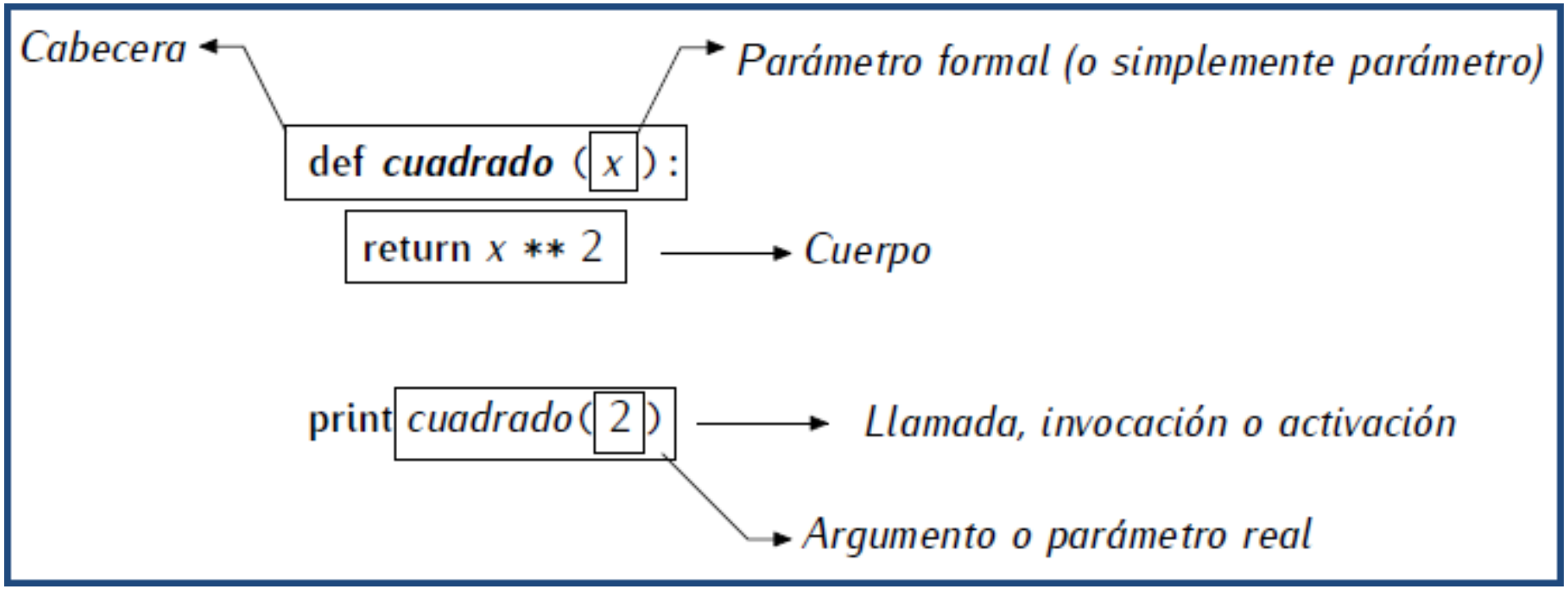
cuadrado.py

```
1 def cuadrado(x):
2     return x ** 2
3
4 print cuadrado(2)
5 a = 1 + cuadrado(3)
6 print cuadrado(a * 3)
```

- La línea que comienza con def, es la **CABECERA de la función.**

- Lo que esta «dentro» de esta instrucción **es el CUERPO de la función**

Definición de Funciones



- Al definir una función, podemos colocar entre «( )» su/s parámetro/s, este se denomina **PARAMETRO FORMAL!**

- El valor que le enviamos al llamar a la función se denomina **PARAMETRO REAL o ARGUMENTO**.

- El código que no este comprendido dentro del cuerpo de la/s función/es. Es el **PROGRAMA PRINCIPAL**.

- **El código del cuerpo de la función solo se ejecutara cuando esta se invoque.**

cuadrado.py

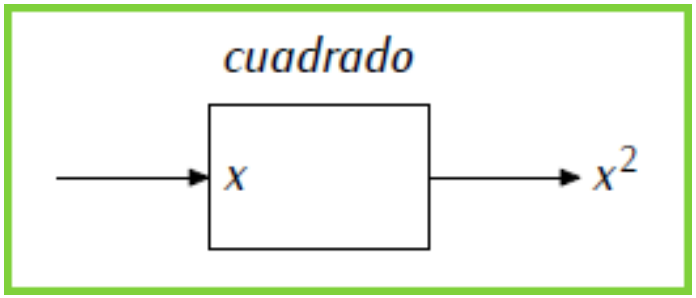
```
1 def cuadrado(x):
2     return x ** 2
3
4 print cuadrado(2)
5 a = 1 + cuadrado(3)
6 print cuadrado(a * 3)
```

- Reglas para dar NOMBRE a las funciones:
- **Similares a las reglas para variables:**

  - **Solo se puede utilizar letras, dígitos y el carácter subrayado.**
  - **La primera letra del nombre NO PUEDE SER UN NUMERO.**
  - **NO se PUEDE usar PALABRAS RESERVADAS.**
  - **NO PUEDE tener el mismo NOMBRE una variable y una FUNCION.**

- Una función, puede verse como un bloque al que se le conecta una entrada y genera una salida.

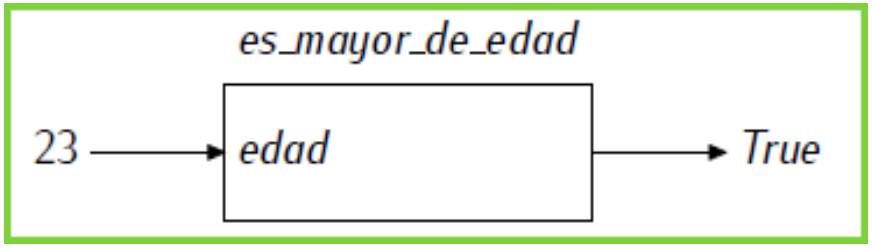
- **Como lo hace, puede no interesarnos.**



- Desde una función se puede invocar a otra función.

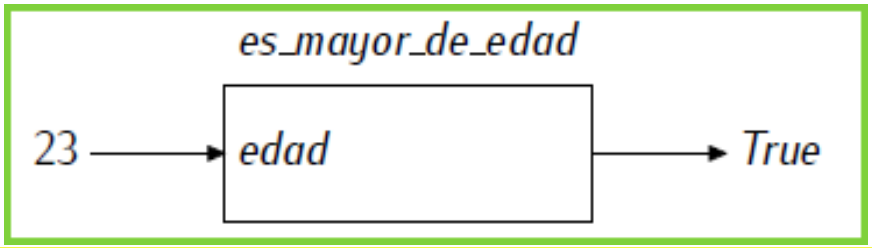
- Adentro del cuerpo de una función, **pueden utilizarse sentencias de selección, iteración**, etc.

- **Y podemos devolver un valor booleano por ejemplo:**




```
1 def es_mayor_de_edad(edad):
2     if edad < 18:
3         resultado = False
4     else:
5         resultado = True
6     return resultado
```

- Una forma usual de trabajar, es utilizar UNA SOLA instrucción RETURN al final de la función. Pero observa el siguiente ej:




```
1 def es_mayor_de_edad(edad):
2     if edad < 18:
3         return False
4     else:
5         return True
```

- También es correcto!. El return fuerza la finalización de la ejecución de la FUNCION.

- Veamos otro ejemplo. Un programa que nos diga si un numero es o no **perfecto.**
- Considerando perfecto al numero que:

  Es igual a la suma de todos sus divisores excluido el mismo.

- Por ejemplo 6. Es divisible por 1, por 2 y por 3. La suma de estos da 6…

- **Comenzamos definiendo la función!**

```
1 def es_perfecto(n):
2     ...
```

- Estamos interesados en conocer TODOS sus divisores. Una vez que tengamos en claro cuales son, solo hay que sumarlos! Si el resultado es el numero: es perfecto!

- Podemos usar un bucle entre 1 y el numero -1, y preguntar si son divisores:

```
1 def es_perfecto(n):
2     for i in range(1, n):
3         if i es divisor de n:
4             ...
```

- Como podemos definir si un numero es divisor de otro?

```
1 def es_perfecto(n):
2     for i in range(1, n):
3         if n % i == 0:
4             ...
```

- Ahora deberíamos sumar todos los elementos que cumplan este IF

```
1 def es_perfecto(n):
2     sumatorio = 0
3     for i in range(1, n):
4         if n % i == 0:
5             sumatorio += i
6     ...
```

- Una vez finalizado el bucle, deberíamos corroborar si la sumatoria es igual al numero y devolver True o False

```
1 def es_perfecto(n):
2     sumatorio = 0
3     for i in range(1, n):
4         if n % i == 0:
5             sumatorio += i
6     if sumatorio == n:
7         return True
8     else:
9         return False
```

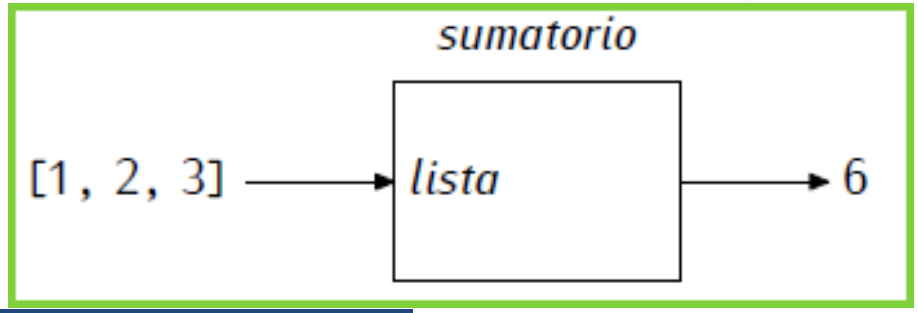
- Observa la siguiente optimización de las ultimas 4 líneas!

```
1  def es_perfecto(n):
2      sumatorio = 0
3      for i in range(1, n):
4          if n % i == 0:
5              sumatorio += i
6      return sumatorio == n
```

- El return, devuelve el resultado de evaluar la expresión lógica de comparar sumatorio y n. True o False!

- Hasta el momento, las funciones recibieron un DATO único como argumento!

- **También podemos crear una función, que reciba como argumento un dato ESTRUCTURADO, una lista!.**




```
1 def sumatorio(lista):
2     s = 0
3     for numero in lista:
4         s += numero
5     return s
```

```
7 a = [1, 2, 3]
8 print sumatorio(a)
```

- Veamos otro ejemplo: un programa que reciba una lista y me devuelva su mayor elemento.

maximo_7.py

```
1 def maximo(lista):
2     for elemento in lista:
3         if elemento > candidato:
4             candidato = elemento
5     return candidato
```

- **Visualiza algún posible error?**

- Cuanto vale candidato en la primer ejecución?...

- Podría darle un valor, suficientemente chico **para que SI o SI todos sean mayores, cual SERIA?**

```
1  def maximo(lista):
2      candidato = lista[0]
3      for elemento in lista:
4          if elemento > candidato:
5              candidato = elemento
6      return candidato
```

La forma correcta es igualarlo al primer elemento. Si es el mayor todo ok, sino será reemplazado posteriormente**.**

- Pero aun hay un error, que pasa si el usuario le envía a esta función **una LISTA VACIA**?

- Intentaremos acceder al elemento [0]. **Generando un INDEX ERROR.**

- Si recibimos una lista vacía, que devolvemos en este caso?.....0?...

- Lo correcto es devolver **NONE**. Este valor es predefinido de python y representa **ausencia de valor**, significa **ninguno.**

- Veamos como quedaría el ejercicio final


```
1  def maximo(lista):
2      if len(lista) > 0:
3          candidato = lista[0]
4          for elemento in lista:
5              if elemento > candidato:
6                  candidato = elemento
7      else:
8          candidato = None
9      return candidato
```

Ahora si, nuestro programa es CORRECTO!

- Hasta el momento, tus programas tienen 3 tipos de líneas.
  - Líneas para importar funciones o módulos.
  - Líneas para definición de funciones.
  - Líneas del programa principal.
- Es importante MANTENER la LEGIBILIDAD de nuestro PROGRAMA.
- Recuerda, el mantenimiento a futuro de nuestro programa es importante y rentable. La legibilidad IMPACTA directamente.

- Ejemplo de ilegibilidad:

```
1  def cuadrado(x):
2      return x**2
3
4  vector = []
5  for i in range(3):
6      vector.append(float(raw_input('Dame un número: ')))
7
8  def suma_cuadrados(v):
9      s = 0
10     for e in v:
11         s += cuadrado(e)
12     return s
13
14 y = suma_cuadrados(vector)
15
16 from math import sqrt
17
18 print 'Distancia al origen:', sqrt(y)
```

- Ejemplo de legibilidad:

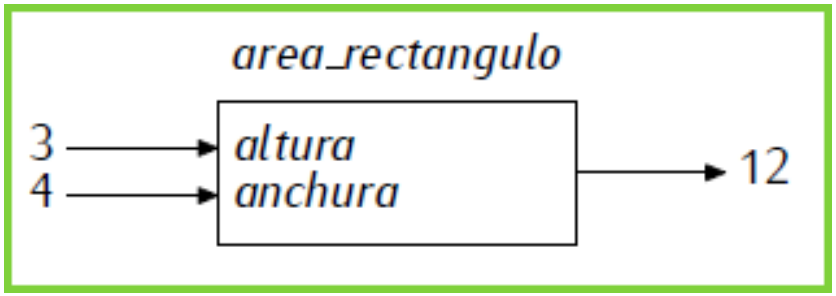
```
1  from math import sqrt
2
3  def cuadrado(x):
4      return x**2
5
6  def suma_cuadrados(v):
7      s = 0
8      for e in v:
9          s += cuadrado(e)
10     return s
11
12 # Programa principal
13 vector = []
14 for i in range(3):
15     vector.append(float(raw_input('Dame un número: ')))
16 y = suma_cuadrados(vector)
17 print 'Distancia al origen:', sqrt(y)
```

Orden:

1-Importar
2-Definir Func.
3-Prog. Ppal.

- 6 Funciones.
  - 6.1 Uso de funciones
  - 6.2 Definición de funciones
    - 6.2.1 Funciones con un solo parámetro
    - 6.2.2 Funciones con varios parámetros
    - 6.2.3 Funciones sin parámetros
    - 6.2.4 Procedimientos: Funciones sin devolución.
    - 6.2.5 Funciones que devuelven LISTAS.

- Una función puede recibir **mas de un parámetro** para producir su resultado.



- Su código:

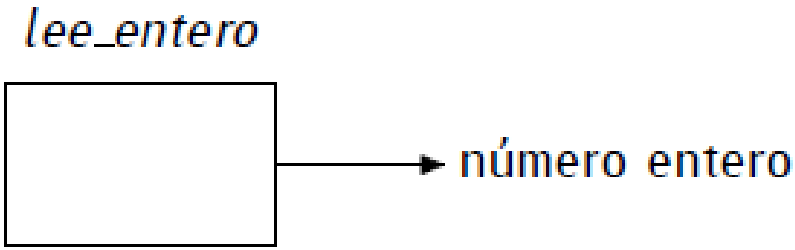
```
1 def area_rectangulo(altura, anchura):
2     return altura * anchura
3
4 print area_rectangulo(3, 4)
```

- 6 Funciones.
  - 6.1 Uso de funciones
  - 6.2 Definición de funciones
    - 6.2.1 Funciones con un solo parámetro
    - 6.2.2 Funciones con varios parámetros
    - 6.2.3 Funciones sin parámetros
    - 6.2.4 Procedimientos: Funciones sin devolución.
    - 6.2.5 Funciones que devuelven LISTAS.

Funciones SIN parámetros

- Es posible definir funciones **SIN parámetros**, PERO es obligatorio el uso de **paréntesis**, en la **definición** y en el **llamado**!...no lo olvides.

- Veamos una función para leer del teclado un VALOR.




```
1 def lee_entero():
2     return int(raw_input())
3
4 a = lee_entero()
```

Funciones sin parámetros

- Los paréntesis son necesarios…veamos el siguiente caso:

```
>>> def saluda():
...     print 'Hola'
...
>>> saluda()
Hola
>>> saluda
<function saluda at 0x8160854>
```

- Aquí al llamar a saluda, sin los paréntesis, el interprete nos responde indicándonos que objetos es y donde esta ubicado.!

Funciones sin parámetros

- Otro uso importante es por ejemplo, desplegar un menú con una función:

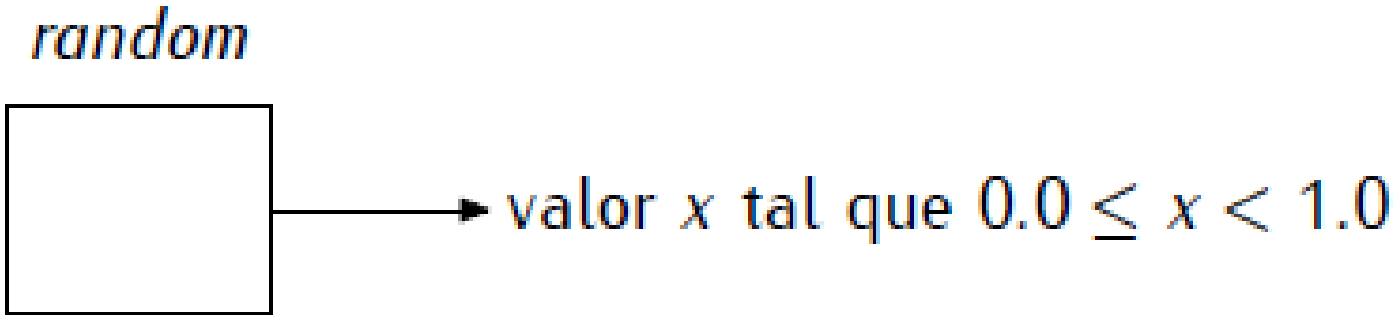
```
1  def menu():
2      opcion = ''
3      while not ('a' <= opcion <= 'c'):
4          print 'Cajero automático.'
5          print 'a) Ingresar dinero.'
6          print 'b) Sacar dinero.'
7          print 'c) Consultar saldo.'
8          opcion = raw_input('Escoja una opción: ')
9          if not (opcion >= 'a' and opcion <= 'c'):
10             print 'Sólo puede escoger las letras a, b o c. Inténtelo de nuevo.'
11     return opcion
```

- Esta función al invocarla, mostrara el menú, y nos DEVOLVERA la opción selecta.!

Funciones sin parámetros

- Hay funciones SIN PARAMETROS que pueden importarse de módulos.

- Una función muy usada es la función RANDOM. (aleatorio)



Funciones sin parámetros

- La función RANDOM del modulo RANDOM, devuelve un numero al AZAR mayor o igual 0.0 y menor que 1.0

```
>>> from random import random
>>> random()
0.7364669743370648 7
>>> random()
0.6416606281483086
>>> random()
0.3633908001684091 9
>>> random()
0.9962223571068339 3
```

- La usaremos mas adelante. Es útil, en estadísticas, etc

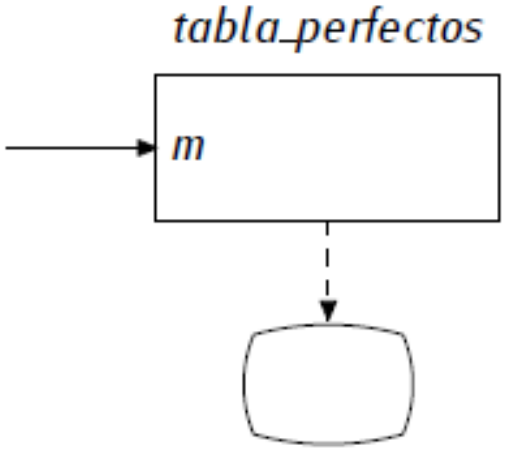
- 6 Funciones.
  - 6.1 Uso de funciones
  - 6.2 Definición de funciones
    - 6.2.1 Funciones con un solo parámetro
    - 6.2.2 Funciones con varios parámetros
    - 6.2.3 Funciones sin parámetros
    - 6.2.4 Procedimientos: Funciones sin devolución de valor.
    - 6.2.5 Funciones que devuelven LISTAS.

- Existen funciones que **NO DEVUELVEN** valor alguno.
- Estas se denominan **PROCEDIMIENTOS**.
- Pueden tener la sentencia RETURN, aunque esta no puede ir acompañada de expresión alguna.
- El significado de RETURN en un procedimiento es indicar el fin de la ejecución de esa función en ese momento.
- Y son útiles? ….veremos a continuación

- Los casos mas comunes de los PROCEDIMIENTOS son los de **INDICAR por pantalla** resultados.

- Esto no implica devolver un valor al programa. Solo **RECIBE argumentos**, OPERA sobre ellos y los **muestra por pantalla**.

- Veamos un programa que haga uso del programa de números PERFECTOS.



- Este programa recibirá un numero, solicitado al usuario, e imprimirá en pantalla todos los números perfectos entre 1 y dicho numero.

```
1  def es_perfecto(n): # Averigua si el número n es o no es perfecto.
2      sumatorio = 0
3      for i in range(1, n):
4          if n % i == 0:
5              sumatorio += i
6      return sumatorio == n
7
8  def tabla_perfectos(m): # Muestra todos los números perfectos entre 1 y m.
9      for i in range(1, m+1):
10         if es_perfecto(i):
11             print i, 'es un número perfecto'
12
13 numero = int(raw_input('Dame un número: '))
14 tabla_perfectos(numero)
```

- Observa que en la función tabla_perfectos, NO HAY SENTENCIA RETURN.

- Si utilizamos el procedimiento como función con devolución que pasa?

```
13 numero = int(raw_input('Dame un número: '))
14 resultado = tabla_perfectos(100)
15 print resultado
```

```
Dame un número: 100
6 es un número perfecto
28 es un número perfecto
None
```

- 6 Funciones.
  - 6.1 Uso de funciones
  - 6.2 Definición de funciones
    - 6.2.1 Funciones con un solo parámetro
    - 6.2.2 Funciones con varios parámetros
    - 6.2.3 Funciones sin parámetros
    - 6.2.4 Procedimientos: Funciones sin devolución de valor.
    - 6.2.5 Funciones que devuelven LISTAS.

Devolución de varios valores

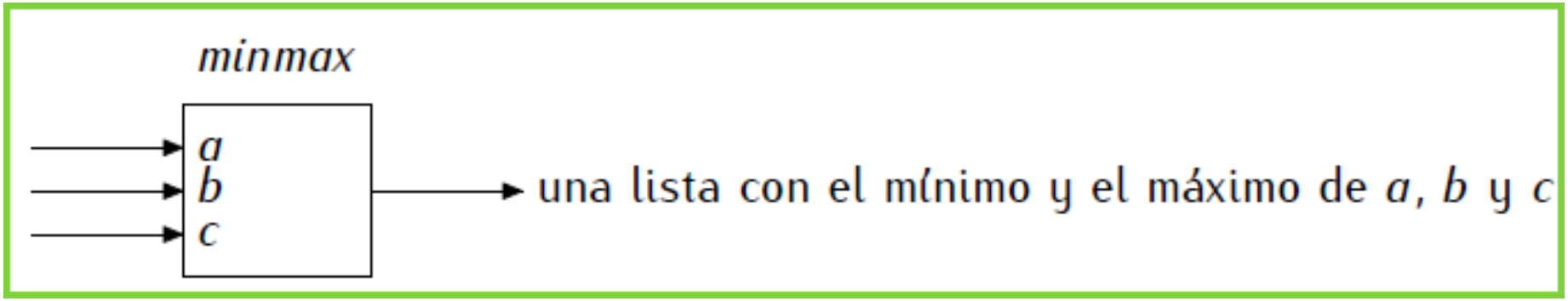
- De acuerdo a lo visto, una función con un return puede devolver un solo valor.

- Las listas, son secuencias de valores, por lo tanto si **devolvemos una lista**, estamos devolviendo mas de un valor.

- Supongamos que nuestro objetivo es crear el siguiente programa:

  - Debe recibir una lista de 3 elementos.
  - Debe devolver el mayor y el menor valor de la lista.

- El problema puede representarse:



Devolución de varios valores

- Para encontrar el mayor y el menor hay muchas formas, veamos una:

```
2     if a < b:
3         if a < c:
4             min = a
5         else:
6             min = c
7     else:
8         if b < c:
9             min = b
10        else:
11            min = c
```

```
12    if a > b:
13        if a > c:
14            max = a
15        else:
16            max = c
17    else:
18        if b > c:
19            max = b
20        else:
21            max = c
```

Devolución de varios valores

- Estas partes armarían nuestra función **minmax**

```
1  def minmax(a, b, c):
2      if a < b:
3          if a < c:
4              min = a
5          else:
6              min = c
7      else:
8          if b < c:
9              min = b
10         else:
11             min = c
```

```
12     if a > b:
13         if a > c:
14             max = a
15         else:
16             max = c
17     else:
18         if b > c:
19             max = b
20         else:
21             max = c
22     return [min, max]
```

Devolución de varios valores

- Veamos que sucede al invocarla:

```
    :
24  a = minmax(10, 2, 5)
25  print 'El mínimo es', a[0]
26  print 'El máximo es', a[1]
```

- Observe la siguiente opción!:

```
    :
24  [minimo, maximo] = minmax(10, 2, 5)
25  print 'El mínimo es', minimo
26  print 'El máximo es', maximo
```

- Cada elemento de la lista a la derecha del igual se asigna a cada elemento en la lista a la izquierda del igual!

Repasando!...

- **Funciones predefinidas!...**
- **Definición de FUNCIONES.**
- **Funciones con UN solo PARAMETRO.**
- **Funciones con VARIOS PARAMETROS.**
- **Funciones SIN PARAMETROS.**

Lo visto!

- **Procedimientos….funciones SIN DEVOLUCION.**
- **Funciones que devuelven varios valores mediante LISTAS.**
- **Todo JUNTO!...**

Ejercicio final

- **Realice un programa, que:**

  1. A través de una función sin parámetros llamada **cargar_datos()** permita al usuario ingresar una lista de 8 elementos. Esta función debe DEVOLVER la lista al programa principal

  2. Implemente un **procedimiento** llamado **mostrar_lista(lista)** que reciba la lista ingresada como argumento y que la muestre en pantalla(no return).

  3. Implemente una función llamada **cantidad_pares(lista)** que reciba como argumento la lista ingresada por teclado y devuelva la cantidad de elementos pares en la misma

Ejercicio final

4. Finalmente su programa deberá implementar una función llamada **lista_pares(lista)**, que reciba la lista original de 8 elementos y genere una nueva lista con los elementos PARES. Esta lista debe ser devuelta al programa principal.

5. Desde el **programa principal** deberá imprimir cantidad de elementos pares y la lista de números PARES. Ambos valores son el resultado de los puntos 3 y 4.