



# **Clase 1:**

# **Introducción a**

# **Python**

# Clase 1 - Temario

- Conceptos Básicos:
  - Variable
  - Tipos de Datos
  - Módulos
  - Funciones
- Estructuras de control:
  - Condicionales: if, if-else, if-elif-else.
  - Iterativas: while y for.
- 

En base a Python 3.1 o version superior

# Antes de empezar...

Python es un lenguaje interpretado. Es un lenguaje Orientado a Objetos pero admite programación imperativa y funcional.

Multiplataforma (Linux, DOS, Windows, etc.)

Con tipado dinámico.

Con gran cantidad de librerías predefinidas.

# Variable

En Python las variables no se declaran. Simplemente, se usan.

```
>>> max = 3
```

```
>>> Max = 2
```

```
>>> max='Ana'
```

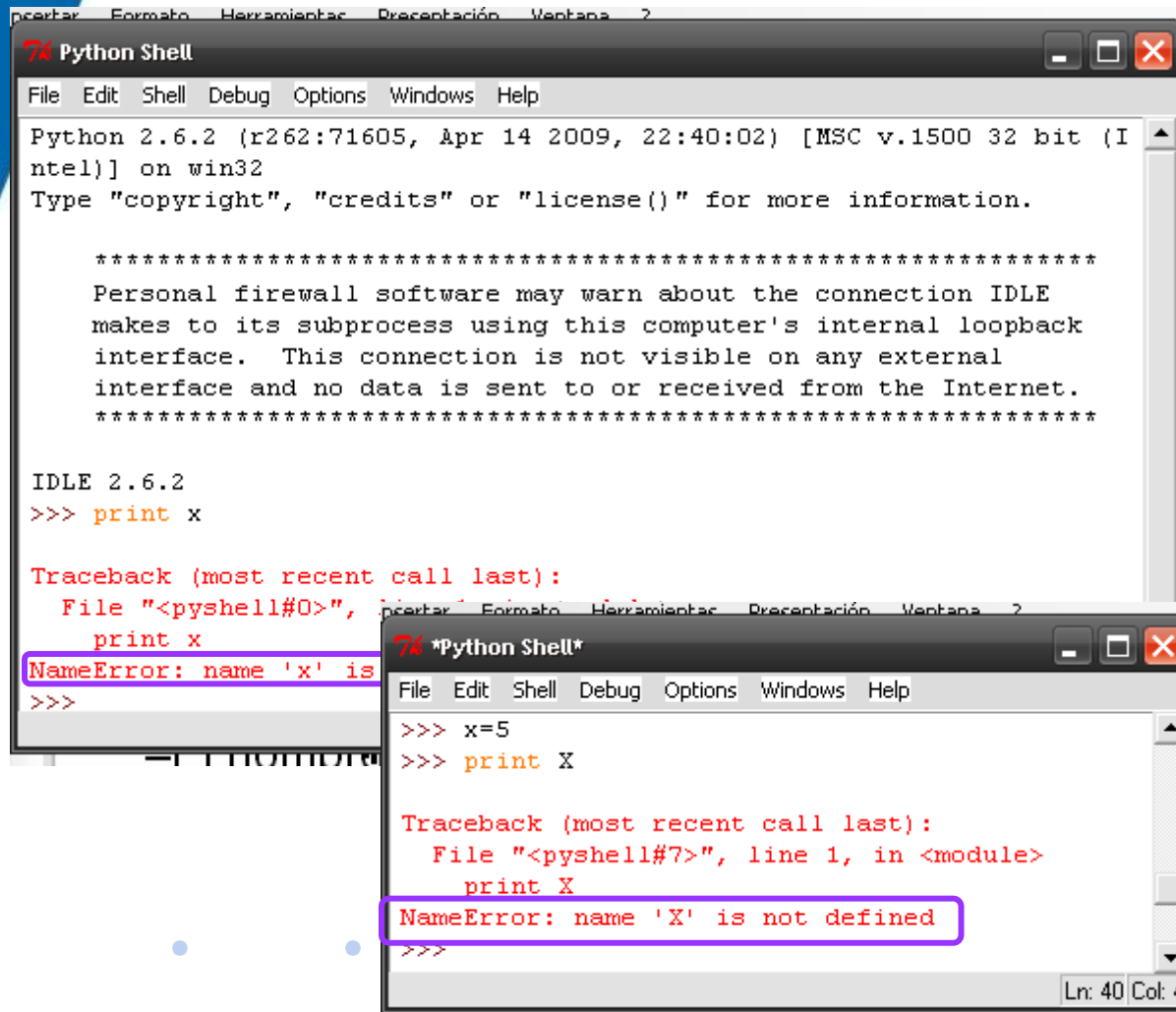
El nombre de las variables pueden contener letras, dígitos y “\_”.  
Deben comenzar con letra.

En Python **HAY** diferencia entre mayúsculas y minúsculas.

## **Importante:**

Hay que asignarle valor a una variable **antes** de poder utilizarla.

# Variable



```
Python Shell
File Edit Shell Debug Options Windows Help
Python 2.6.2 (r262:71605, Apr 14 2009, 22:40:02) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.

*****
Personal firewall software may warn about the connection IDLE
makes to its subprocess using this computer's internal loopback
interface. This connection is not visible on any external
interface and no data is sent to or received from the Internet.
*****

IDLE 2.6.2
>>> print x

Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    print x
NameError: name 'x' is not defined
>>>
```

```
*Python Shell*
File Edit Shell Debug Options Windows Help
>>> x=5
>>> print X

Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    print X
NameError: name 'X' is not defined
>>>
```

**Ej. de uso de variable sin un valor previo**

**Ej. de uso de mayúscula y minúscula**

# Variable

- `x=3`      `nom='pepe'`      → asignación
- `print ('ingrese una edad')`
- `edad=input()`      → edad es una variable
- `edad=input('ingrese una edad')`
- Lo ingresa como string por lo tanto hay que convertirlo luego.
- `edad=int(edad)`
- Para ingresar una cadena de caracteres con input se escriben los datos de entrada en la consola sin usar comillas
- `print('ingrese su nombre')`
- `name=input()`

# Recordemos: Qué es un Tipo de datos?

## Definición:

Un Tipo de datos define un conjunto de valores y las operaciones válidas que pueden realizarse sobre esos valores

### —Conjunto de valores:

- Representa **todos los valores posibles** que puede llegar a tomar una variable de ese tipo

### —Operaciones permitidas:

- Establece qué operaciones son válidas para los datos pertenecientes a dicho tipo

# Tipos Básicos - Enteros

Al asignar un número a una variable, le asociará el tipo “**int**” en caso que su valor entre en 32 bits, caso contrario reserva automáticamente para un “**long**”.

Miremos este caso:

**Entero**

**Entero largo**

```
>>> x=2147483647
>>> type(x)
<type 'int'>
>>> x= x+1
>>> type(x)
<type 'long'>
>>> |
```

¿Qué  
pasó?

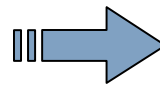


# Tipos Básicos - Reales

- Permite trabajar con valores con coma decimal.
- Se representan mediante el tipo **float**.
- Se almacenan en 64 bits.
- El rango de valores es de:
  - $\pm 22250738585072020 \times 10^{-308}$  a
  - $\pm 17976931348623157 \times 10^{308}$

```
>>> var_real1= 0.2703
```

```
>>> var_real2= 0.1e-3
```



**Notación científica. Equivale al número:**

$$0.1 \times 10^{-3} = 0.1 \times 0.001 = 0.0001$$

Para el caso de necesitar representar fracciones de forma más precisa, se cuenta con el tipo **decimal**, desde la versión 2.4 de Python

# Operadores aritméticos

- Operaciones que pueden hacerse sobre variables numéricas y números.

Operadores aritméticos	Operador	Descripción
	+	Suma
	-	Resta
	*	Multiplicación
	/	División
	-	Negación
	**	Exponente
	//	División entera
	%	Resto de la división

# Tipos Básicos - Booleanos

- Se los utiliza para indicar valores de verdad
- Permiten dos únicos valores:
  - True
  - False
- Operadores lógicos: and, or, not

<b>and</b>	True	False
True	<b>True</b>	False
False	False	False

<b>or</b>	True	False
True	True	True
False	True	<b>False</b>

<b>not</b>	
True	False
False	True

# Tipos Básicos - Booleanos

- Operadores relacionales: ==, !=, <, <=, >, >=
- Supongamos x=2 e y=3

Operador	Descripción	Ejemplo	Resultado
==	¿x es igual a y?	x==y	<b>False</b>
!=	¿x es distinto a y?	x!=y	<b>True</b>
<	¿x es menor que y?	x<y	<b>True</b>
>	¿x es mayor que y?	x>y	<b>False</b>
<=	¿x es menor o igual que y?	x<=y	<b>True</b>
>=	¿x es mayor o igual que y?	x>=y	<b>False</b>

# Tipos Básicos - Cadenas

- No todos son números....
- Usamos cadenas de caracteres para valores que representan:
  - Nombres de personas, países, ciudades
  - Direcciones postales, de mail,
  - Mensajes,
  - Etc.
- Ejemplos:
- “Juan Pérez”; “Argentina”; “calle 60 y 124”;  
["juan.perez@gmail.com"](mailto:juan.perez@gmail.com), “Hola que tal”

# Tipos Básicos - Cadenas

Secuencia de caracteres (letras, números, marcas de puntuación, etc.)

Se encierran entre comillas simples ' ' o comillas dobles “ ”

Algunos operadores: **+ Concatenación**

**\* Repetición**

```
>>> nombre='Pepe '
```

```
>>> apellido="Lopez"
```

```
>>> nombre + apellido
```

```
'Pepe Lopez'
```

+ operador de concatenación entre dos cadenas

```
>>> 'Lopez' *5
```

```
'LopezLopezLopezLopezLopez'
```

\* operador de repetición de cadenas

# Tipos Básicos - Cadenas

Operadores de comparación: ==, !=, >, <, >=, <=

Ejemplos:

```
>>> 'pepe' == 'pepe'
```

```
true
```

```
>>> "juan" < "ana"
```

```
false
```

Python utiliza un criterio de comparación de cadenas muy natural: **el orden alfabético**.

Utiliza los códigos ASCII de los caracteres para decidir su orden.

# Tipos Básicos - Cadenas

Funciones predefinidas que manipulan cadenas:

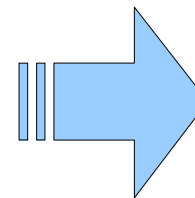
Funciones	Descripción	Ejemplo	Resultado
<b>int</b>	Convierte una cadena numérica a un valor entero	<code>int("123")</code>	123
<b>float</b>	Convierte una cadena numérica a un valor real	<code>float("123")</code>	123.0
<b>ord</b>	Devuelve el código ASCII de la cadena	<code>ord("a")</code>	97
<b>chr</b>	Devuelve el carácter correspondiente al valor ASCII	<code>chr(89)</code>	"T"
<b>str</b>	Convierte un valor entero a una cadena de caracteres	<code>str(123)</code>	"123"



# Tipos Básicos - Cadenas

Para saber el orden que ocupa un carácter se cuenta con las funciones predefinidas “**ord()**” y “**chr()**”, su función inversa.

```
>>>  
>>> ord('a')  
97  
>>> chr(78)  
'N'  
>>> |
```



Notar  
que:  
'N' < 'a'!!!!

# Tipos Básicos - Cadenas

Otras cosas útiles.... Si `a` es una variable de tipo cadena se le puede aplicar:

Funciones	Descripción	Ejemplo	Resultado
<b>a.lower</b>	Convierte los caracteres de la cadena <code>a</code> a minúsculas	<code>a="HOLA"</code> <code>print a.lower</code>	<code>"hola"</code>
<b>a.upper</b>	Convierte los caracteres de la cadena <code>a</code> a mayúsculas	<code>a="hola"</code> <code>print a.upper</code>	<code>"HOLA"</code>

# Tipos Básicos – Cadenas. Otras funciones

a.isupper()

a.islower()

a.isdecimal() → si es nro y cadena no vacía

a.isalpha() → si es letra y cadena no vacía

a.isalnum() → numero y letras, cade.no vacía

a.isspace() → espacio

a.istitle() → título, 1er letra may resto minus

a.startswith('s')

a.endswith('z')

m.split() → retorna la lista de palabras que conforman a la cadena m que están separadas por blancos

# Tipos Básicos - Cadenas

## Longitud de las cadenas

- Uso de función predefinida **len()**

```
>>> cadena = 'Hola que tal'
>>> print('La longitud de la cadena es: ', len(cadena))
('La longitud de la cadena es: ', 12)
```

**len("")** devuelve longitud **0**

**len(' ')** devuelve longitud **1**

# Tipos Básicos - Cadenas

Accediendo a los caracteres de las cadenas

**cadena = 'Hola que tal'**

0	1	2	3	4	5	6	7	8	9	10	11
H	o	l	a		q	u	e		t	a	l

```
>>> cadena = 'Hola que tal'
>>> print cadena[0]
H
>>> print cadena[5:8]
que
>>> print cadena[9:-1]
ta
>>>
```

El operador `:` (slicing), nos permite obtener subcadenas.

`[:]` devuelve toda la cadena

Indices negativos, recorren de derecha a izquierda la cadena

# Tipos Básicos - Cadenas

Cadenas que ocupan más de una línea: Uso de '''

```
>>> print '''Este string ocupa  
dos líneas'''
```

```
Este string ocupa  
dos líneas
```

```
>>> print 'Este string ocupa
```

```
SyntaxError: EOL while scanning string literal
```

```
>>> |
```

**Cadenas que ocupan  
más de una línea**



# Estructuras de Control

# Decisiones ....

**Sentencias condicionales:** Permiten comprobar condiciones y que el programa se comporte de una manera u otra, de acuerdo a esa condición.

```
if  
if .. else  
if .. elif.. elif.. else
```



# Sentencia if

**Sentencia if:** Sentencia condicional más simple. Permite tomar decisiones sencillas. Solo rama V:

Condición

```
if x > 0:  
    print ("x es positivo")
```

La **indentación** indica que esas sentencias deben ejecutarse si la **condición** se cumple. Se deben dejar 4 espacios en blanco.

# Sentencia if

## Ejemplo

Atención a la indentación

```
>>>  
>>> numero=1  
>>> if numero==1:  
    print "Estamos mostrando una sentencia if con varias lineas"  
    print "En este caso, estas lineas se imprimen si la variable numero es 1"  
    print "Luego de imprimir estas lineas, pasmaos a la siguiente instruccion"
```

```
Estamos mostrando una sentencia if con varias lineas  
En este caso, estas lineas se imprimen si la variable numero es 1  
Luego de imprimir estas lineas, pasmaos a la siguiente instruccion  
>>>  
\\
```

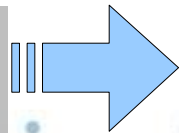
# Decisiones ....

## Sentencia if-else

Permite establecer las acciones por rama V y por rama F.

Ejemplo:

Operador %  
(módulo)



```
if x%2 == 0:  
    print( x, "es par")  
else:  
    print (x, "es impar")
```

# Decisiones ....

## Sentencia if-elif

¿Qué pasa cuando hay más de dos condiciones?

Ejemplo:

**If edad >= 0 and edad <2:**

**print("bebe")**

**elif edad >=2 and edad <13:**

**print ("niño/a")**

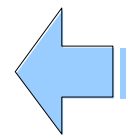
**elif edad >=13 and edad <20:**

**print ("adolescente")**

**else:**

**print("adulto")**

Aquí aparecen  
varias  
condiciones  
excluyentes.



¿A qué valores  
referencia el  
else?

# Iteraciones

**Bucles:** Permiten ejecutar cierto código un número reiterado de veces hasta que se cumpla una condición.

Python tiene dos sentencias iterativas:

while

for .. in

# Iteramos con for

Forma general:

```
for variable in lista de Valores:  
    sentencias
```

for j in range (inicio, tope, paso):

sent1  
sent2

La variable j toma todos los valores que aparecen en la lista hasta (tope -1) y luego termina la iteración

# Iteramos con for

- **Función range():** Devuelve una lista de números enteros.

Formas de usarla:

1 argumento:

**range(5)**, devuelve **[0,1,2,3,4]** - Desde 0 hasta el argumento – 1

2 argumentos:

**range(2,5)**, devuelve **[2,3,4]** - Desde el arg.1 hasta el 2do arg. – 1

3 argumentos:

**range(2,5,2)**, devuelve **[2,4]** Desde el arg.1 hasta el 2do arg. – 1, pero con un incremento de 2)

# Iteramos con for

Formas de usarla:

```
suma=0
```

```
for j in range(0,5):
```

```
    suma= suma + j
```

```
print("la suma es:", suma) → imprime 10
```



# Sentencia while

Forma general:

```
while condición:  
    sentencias
```

- La condición se evalúa cada ciclo, y mientras sea verdadera, la iteración continúa.
- Importante: La condición DEBE hacerse falsa en algún momento. ¿Qué pasa si esto no sucede?

# Sentencia while

Ejemplo de uso:

```
cont=0
```

```
val=int(input("ingrese un valor, 0 finaliza"))
```

```
while val != 0:
```

```
    if val%2==0:
```

```
        cont=cont + 1
```

```
    val=int(input("ingrese un valor, 0 finaliza"))
```

```
print("la cantidad de valores pares es", cont)
```

# While vs for

- Ambas son sentencias iterativas
- En ambas sentencias, las acciones ejecutadas en el bucle deben estar indentadas
- Diferencia:
  - La sentencia while evalúa una condición que debemos asegurarnos se haga falsa en algún momento
  - La sentencia for, itera un número fijo de veces: hasta que la variable tome todos los posibles valores de la lista.

# Listas

- Colección ordenada de datos
- Puede contener **cualquier tipo de datos**, inclusive listas.
- Ejemplos

`Lista1=[]`

`Lista2=[1,2,3]`

`Lista3=[1, "Hola"]`

`Lista4= [22, True, 'una lista', [1,7]]`

# Listas

## ¿Cómo accedemos a los elementos de la lista?

- Indicar el índice del elemento (posición dentro de la lista), entre corchetes `[]`.
- Ejemplo:
  - `print (Lista2[2])`
  - `Lista4[1] = False`, esto provoca que el **2do** elemento de la lista se cambie  
`Lista4= [22, False, 'una lista', [1,7]]`
- **IMPORTANTE**: los índices comienzan en **0**.

# Listas

Lista4= [22, True, 'una lista', [1,7]]

- Para **acceder** a elementos que son “listas”, se debe usar también **[]**. El primero indica posición de la lista exterior, los otros indican posición de las listas interiores. Ej.: **Lista4[3][1]**, devuelve **7**
- Se pueden usar índices negativos. En ese caso se comienza a contar desde atrás. Ej.: **Lista4[-3]**, devuelve **True**

# Listas

```
lis1 = [22, True, 'una lista', [1,7]]
```

	Descripción	Ejemplo	Resultado
<b>append</b>	Agrega un elemento al final de la lista	<code>lis1.append(4)</code>	<code>[22, True, 'una lista', [1,7], 4]</code>
<b>count</b>	Cuenta la aparición de un elemento de la lista	<code>lis1.count(22)</code>	<b>1</b>
<b>index</b>	Devuelve la posición de un elemento dentro de la lista	<code>lis1.index('una lista')</code>	<b>2</b>
<b>del</b>	Elimina un elemento	<code>del lis1[2]</code>	<code>[22, True, [1,7]]</code>

# Listas

```
lis1= [22, True, 'una lista', [1,7]]
```

Pertenencia → True in lis1

lis1.insert(1,24) → **lis1= [22, 24, True, 'una lista', [1,7]]**

lis1.remove('una lista') → si no existe el elemento da error

lis1.sort() → debe ser homogénea

lis1.sort(reverse=True)



# Listas

```
lis1 = [22, True, 'una lista', [1,7]]
```

## Slicing:

Permite seleccionar **porciones** de listas:

Para seleccionar parte de una lista se debe colocar dos índices: **inicio:fin**. Indica que queremos la parte de la lista que comprende desde el índice inicio hasta el elemento anterior a **fin**. **NO** incluye al elemento cuyo índice es fin.

Ej.: `lis1[1:3]`, devuelve la lista `[True, 'una lista']`

Si no se pone inicio o fin, se toma por defecto las posiciones de inicio y fin de la lista.

Ej.: `lis1[:2]`, devuelve la lista `[22, True]`

`lis1[2:]`, devuelve la lista `['una lista', [1,7]]`

# Listas

```
>>>
```

```
>>> long=5  
>>> cadena='lista'  
>>> lista=['esto es un',cadena,'de',long,'elementos']  
>>> print(lista)  
['esto es un', 'lista', 'de', 5, 'elementos']  
>>> print( lista[0])  
esto es un  
>>> print(lista[1:-2])  
['lista', 'de']  
>>> print(lista[2:-4])  
[]
```

# Tuplas

¿ Qué es una tupla?

- Son colecciones de datos ordenados.
- Se definen de la siguiente manera:

**Tupla1=1,2   ó   Tupla1=(1,2)**

- Se parecen a las listas sólo que son INMUTABLES:  
No se las puede modificar!

```
>>> tupla=(2,4)
>>> tupla[0]=6

Traceback (most recent call last):
  File "<pyshell#30>", line 1, in <module>
    tupla[0]=6
TypeError: 'tuple' object does not support item assignment
>>>
```

# Tuplas

## ¿Cómo accedemos a los elementos de la tupla?

Similar a las listas....se usan los `[]`

Ej.:

`tupla1=(1,2)`

`tupla1[1]` devuelve 2

Las tuplas pueden contener elementos mutables, ej. listas

```
>>> tupla=(1, 'hola', [2,3])
>>> tupla[2][0]= 'cambie'
>>> tupla
(1, 'hola', ['cambie', 3])
```

`tupla[2]` es una lista, tipo mutable, por lo tanto se puede cambiar el contenido de la lista

# Módulos

- Los **módulos** son archivos de texto cuya extensión es: .py, .pyc o .pyw
- Contienen instrucciones Python. Se pueden definir funciones dentro del módulo
- Se los puede ejecutar cuantas veces se quiera
- Se los puede importar y usar desde otros módulos

# Módulos: Sentencia import

¿Qué significa “importar” un módulo?

```
import nombre_modulo
```

- Se tiene “acceso” a las funciones definidas dentro del módulo.
- Para invocar a estas funciones:

```
nombre_modulo.nombre_funcion
```

También se puede usar:

```
from nombre_modulo import nombre función/es
```

# Funciones

```
def nombreFuncion(parametros):  
    sentencias  
    return <expresion>
```

El cuerpo de la función **debe** estar indentado!

## Ejemplo sencillo:

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

## ¿Cómo lo usamos?

```
print (cuadrado(3))  
a=2+cuadrado(9)
```

x es el parámetro de la función

# Pasaje de parámetros

```
def nombreFuncion(parametros):
```

```
    sentencias  
    return <expresion>
```

El pasaje de parámetros puede ser por copia o referencia.  
No se lo identifica con ninguna palabra el tipo de pasaje.

- Si el parámetro es inmutable, va por copia (ej. una constante o una tupla)
- Si el parámetro es mutable, va por referencia (ej. una lista)



# Pasaje de Parámetros

Los parámetros pueden pasarse respetando cantidad y posición.(\*1)

Pero Python admite omitir alguno (\*2), definir en la llamada la relación entre el actual y el formal(\*3), o usar una cantidad variable de parámetros.

# Pasaje de Parámetros por copia

```
def suma(x,y):  
    return x + y
```

¿Cómo lo usamos?

```
print (suma(3,4))    (*1)
```

```
def pot(x,y=2):  
    return x **y
```

¿Cómo lo usamos?

```
print (pot(3))    (*2)
```

→ imprime 9

```
def suma(x,y):  
    return x + y
```

¿Cómo lo usamos?

```
print (suma(y=5, x=4))    (*3)
```

# Pasaje de Parámetros por referencia

```
def aumento(suel,cant):  
    suel[0]=suel[0] + cant
```

**¿Cómo lo usamos?**

```
sueldo=[]
```

```
sueldo.append(500)
```

```
aumento(sueldo,20) → sueldo es una lista, por lo tanto es mutable. Se asume  
pasaje por referencia
```

```
print (sueldo[0])
```

# Variables locales y globales

```
x=12  
a=13
```

Variables globales

```
def mi_funcion(a):
```

```
→ x=9  
→ a=10
```

Variables locales

**Variables locales enmascaran las globales**

# Variables locales y globales

```
x=12  
a=13
```

Variables globales

Acceso a las variables globales mediante **global**

```
def miFuncion(a):
```

```
    global x
```

Variables locales

```
    x=9
```

Acá usa la variable global x

```
    a=10
```

# Ejemplo integrador

Sumar los elementos de una lista dada y contar los que son positivos.

```
def suma(lis):
```

```
    s=0
```

```
    for j in range (0, len(lis)):
```

```
        s= s + lis[j]
```

```
    suma= s
```

```
    return suma
```

```
c=0
```

```
li=[]
```

```
num=int(input("ingrese un nro, 0 finaliza"))
```

```
while num !=0:
```

```
    li.append(num)
```

```
    if num > 0:
```

```
        c=c+1
```

```
    num=int(input("ingrese un nro, 0 finaliza"))
```

# Ejemplo integrador

```
print("la suma de los elementos de la lista es:", suma(li))  
input()
```

```
print("la cantidad de valores positivos es:", c)  
input()
```

```
• • • • • • • • • •  
• • • • • • • • • •  
• • • • • • • • • •
```

```
• • • • • • • • • •
```

