# Lenguaje de Programación Python

## Funciones

Dr. Mario Marcelo Berón

Universidad Nacional de San Luis
Departamento de Informática
Área de Programación y Metodologías de Desarrollo de Software

# Actores

## Sr. Py

Expondrá los conceptos a lo largo de esta clase.

## Sr. Boa

Presentará los ejemplos y ayudas.

## Sra. Anaconda

Presentará las situaciones propensas a errores.

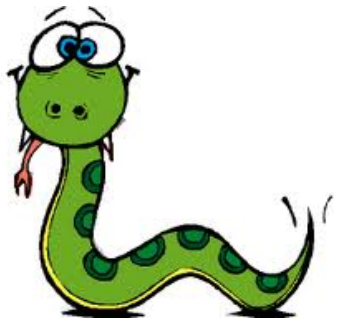
# Motivación

Las funciones son muy importantes para la resolución de problemas porque:

- Dividen tareas grandes en otras más pequeñas.
- Permiten construir sobre lo que otros ya han hecho.
- Ocultan detalles de implementación que no se necesitan saber.
- Permiten realizar la misma tarea sobre conjuntos distintos de datos.
- Hacen posible modularizar el problema, de esta manera cada subtarea puede ser realizada por una función.

En esta clase vamos a introducir los conceptos necesarios para la definición y uso de funciones.

# Conceptualización



## Concepto

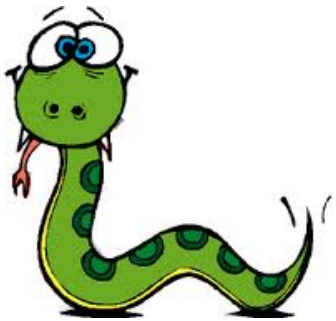
Segmento de código que realiza una actividad específica y retorna un resultado.

## Partes

Las funciones tienen dos partes bien definidas:

- La Definición
- La Invocación

# La Definición



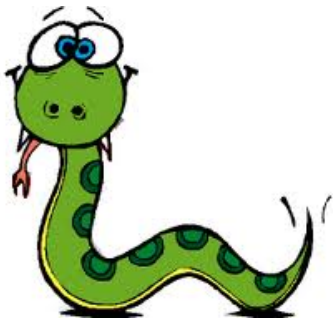
## Sintaxis

def <Ident.> ( <Lista de Par. Formales> ) Cuerpo

## Partes de la Defínción

- Ident: es el nombre de la función (un identificador).
- Lista de Parámetros Formales: es un mecanismo que provee los datos requeridos por la función para realizar su tarea.
- Cuerpo: especifica la tarea que va a realizar la función. Normalmente consta de un conjunto de proposiciones debidamente indentadas.

# Parámetros Formales



## Sintaxis

def <Ident.> ( <**Lista de Par. Form.**> ) Cuerpo

## Concepto

Variables que tomarán un valor cuando se invoque la función. Los parámetros formales se utilizan sólo dentro del cuerpo de la función.

# Parámetros Formales

## Sintaxis

def <Ident.> ( <**Lista de Par. Form.**> ) Cuerpo

## Importante

- Una función puede no tener parámetros formales.
- No se realiza chequeo de tipo de los argumentos, esto útil para sobrecargar sin esfuerzo.

## Cuidado

La función no ejecuta correctamente si no se pasan los argumentos adecuados.

# Parámetros Formales

## Sintaxis

def <Ident.> ( <**Lista de Par. Form.**> ) Cuerpo

## Pasaje de Parámetros

- Cuando se pasa un parámetro en Python lo que en realidad se pasa es una copia de la referencia a un objeto (Similar a lo que un programador hace cuando quiere simular los parámetros por dirección cuando programa en Lenguaje C).
- En Python las modificaciones que se realizan sobre los parámetros no siempre repercuten en el exterior porque depende si el objeto es:
  - Mutable.
  - Inmutable.

# Parámetros Formales

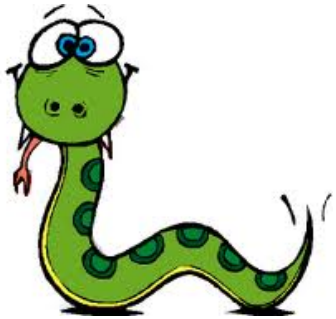
## Pasaje de Parámetros

- Objeto Mutable: los cambios realizados en el cuerpo de la función repercuten en el exterior.
- Objeto Inmutable: los cambios realizados en el cuerpo de la función no repercuten en el exterior.

## Lección Práctica

- Los parámetros que se corresponden con objetos inmutables se comportan como si su pasaje fuese por valor.
- Los parámetros que se corresponden con objetos mutables se comportan como si su pasaje fuese por dirección.

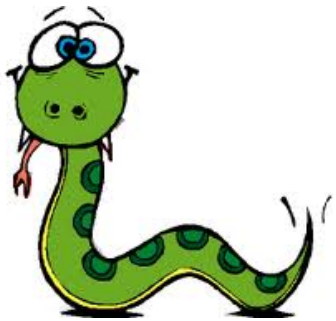
# Parámetros Formales



## Funciones con Argumentos con Valores por Defecto

- Los argumentos pueden tener valores por defecto.
- Cuando un argumento no es provisto en la llamada se usa el valor por defecto
- Si un argumento de una función no tiene valores por defecto y dicho argumento no está provisto en la llamada se produce un error.
- Se pueden usar nombres explícitos para cambiar el orden de los argumentos.

# Cuerpo



**Sintaxis**

def <Ident.> ( <Lista de Par. Form.> ) **Cuerpo**

**Cuerpo**

Es un conjunto proposiciones debidamente indentadas que implementan la tarea que la función debe realizar.

# Cuerpo

## Devolucón de un Valor

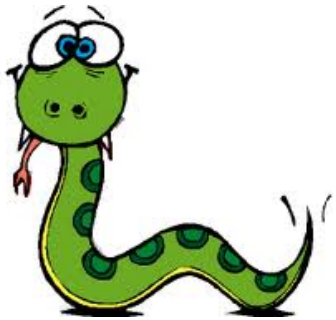
Una función devuelve un único valor. Para realizar dicha tarea se utiliza la proposición **return**, cuya sintaxis es la siguiente:

return $<$ *expr* $>$

## Cuidado

- Si una función no contiene proposiciones **return** se ejecuta hasta que llegue al final del cuerpo y el valor retornado es **None**.
- Se puede colocar una sentencia **return** sin valor de retorno, en este caso nuevamente el valor retornado es **None**.

# Cuerpo



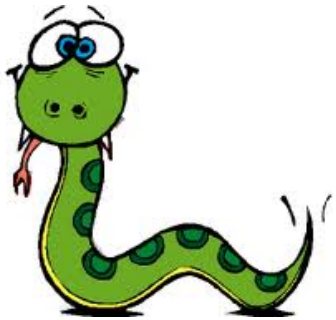
## Sintaxis

def <Ident.> ( <Lista de Par. Form.> ) **Cuerpo**

## Variables Locales

Dentro del cuerpo de las funciones es posible crear variables. Dichas variables son sólo accesibles dentro del cuerpo de la función y por esta razón reciben el nombre de variables locales. El tiempo de vida de estas variables comienza cuando la función se ejecuta y finaliza cuando la función termina su ejecución.

# Invocación de Funciones



## Sintaxis

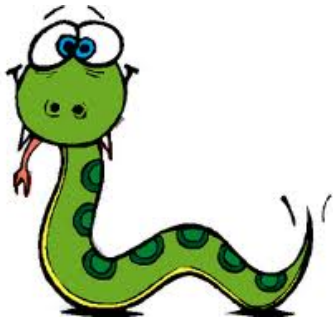
<Ident.> ( <Lista de Parámetros Reales> )

## Invocación

- Para que una función realice la tarea para la cual fue creada la misma debe invocarse.
- Las funciones se invocan desde el cuerpo de otra función o bien desde otras partes del programa.
- Para invocar una función se debe:

  Colocar el nombre de la función seguido por la lista de **Parámetros Reales** encerrados entre paréntesis.

# Invocación de Funciones



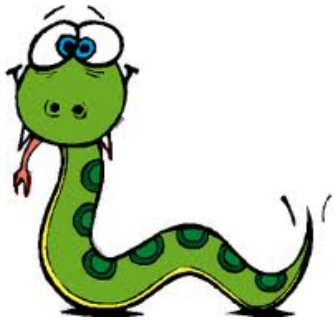
## Sintaxis

<Ident.> ( <Lista de Parámetros Reales> )

## Invocación

- Cuando una función se invoca el control del programa se pasa al cuerpo de la función.
- Cuando una función finaliza su ejecución (por medio de una proposición **return** o cuando se ejecutó la última proposición de su cuerpo) el control del programa regresa a la expresión que se encuentra después de la invocación.

# Invocación de Funciones



## Sintaxis

<Ident.> ( <Lista de Parámetros Reales> )

## Correspondencia de Parámetros

- La correspondencia entre los Parámetros Formales y los Parámetros Reales es Posicional. Es decir, el primer parámetro real se corresponde con el primer parámetro formal, el segundo parámetro real con el segundo parámetro formal y así siguiendo.
- También se puede:
  - Obviar parámetros siempre y cuando la definición de la función provea valores por defecto.
  - Cambiar el orden de los parámetros colocando la asignación explícita en la invocación.

# Parámetros Formales

## Ejemplo

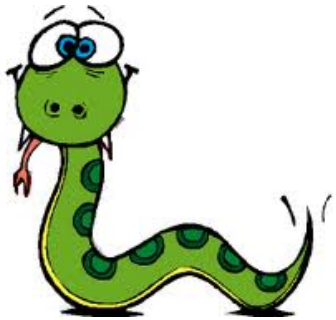
```
def default_args(a, b='bar', c=13):
    return "a=%s, b=%s, c=%s" % (a, b, c)


# usa todos los valores por defecto
print default_args('apa')

# usa un valor por defecto
print default_args('s', b='py')

# cambia el orden
print default_args(c=26, a='apa')
```

# Invocación de Funciones



## Sintaxis

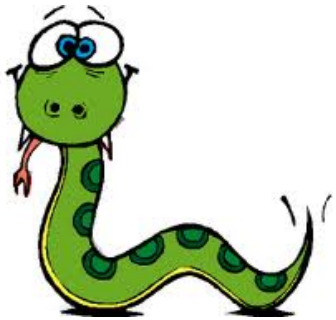
<Ident.> ( <Lista de Parámetros Reales> )

## Parámetros Reales

Los Parámetros Reales pueden ser:

- Variables.
- Constantes.
- Expresiones más complejas.

# Invocación de Funciones



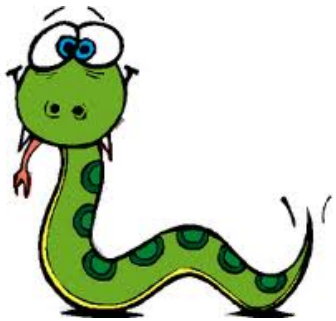
## Sintaxis

<Ident.> ( <Lista de Parámetros Reales> )

## Invocación

- Generalmente cuando una función devuelve un valor suele estar dentro de expresiones.
- Cuando una función no devuelve un valor la llamada a la función aparece sola.

# Ámbitos de las Variables

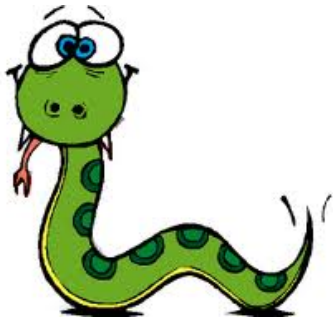


## Ámbito de las Variables

Las variables se pueden clasificar de acuerdo a su ámbito de referencia (la parte del programa donde la variable es reconocida) en:

- Locales.
- Globales.

# Ámbitos de las Variables



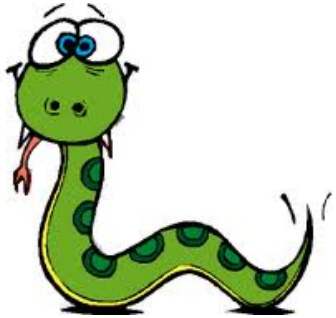
## Variables Locales

Cualquier variable que se cree dentro de una función es local a la misma. En otras palabras el ámbito de referencia es dicha función.

## Importante

La característica mencionada previamente permite que existan variables con el mismo nombre en diferentes funciones. Dichas variables no mantienen ninguna relación entre si.

# Ámbitos de las Variables



## Variables Globales

El ámbito de esta clase de variables se extiende desde el punto de su creación hasta el final del programa. Las variables globales se crean fuera de las funciones. Si una función desea utilizar una variable global lo puede hacer por medio del nombre de la misma.

# Más sobre Funciones

- Funciones con chequeo de tipos explícito.
- Funciones Anidadas.
- Funciones Recursivas.
- Funciones de Orden Superior Primitivas y no Primitivas.
- Funciones con un Número Variable de Argumentos.