Funciones, Subrutinas o Procedimientos

Jhon Jairo Padilla Aguilar, PhD.

Definición de función

- En el contexto de la programación, una función es una secuencia de sentencias que realizan una operación y que reciben un nombre.
- Cuando se define una función, se especifica el nombre y la secuencia de sentencias. Más adelante, se puede “llamar” a la función por ese nombre.

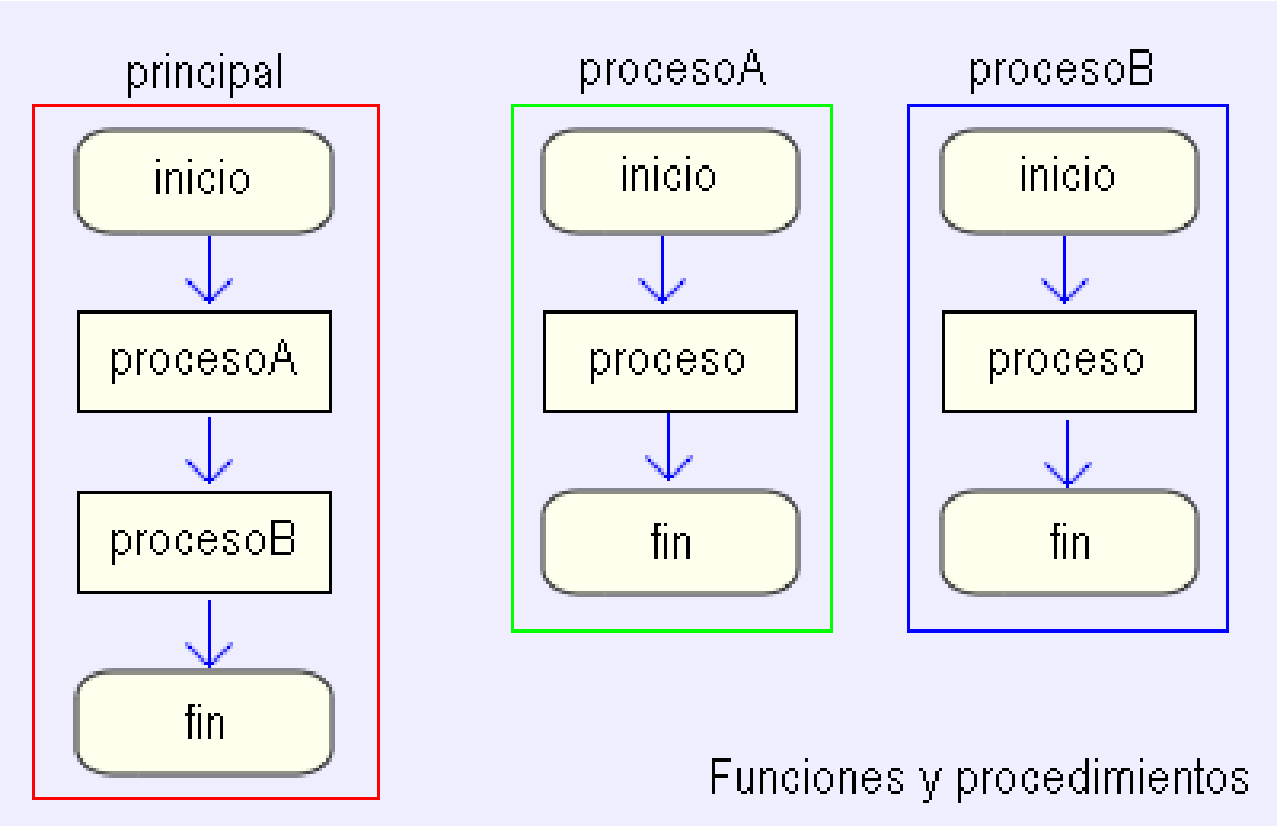
Ejemplo de una función

- Ya hemos visto un ejemplo de una llamada a una función:

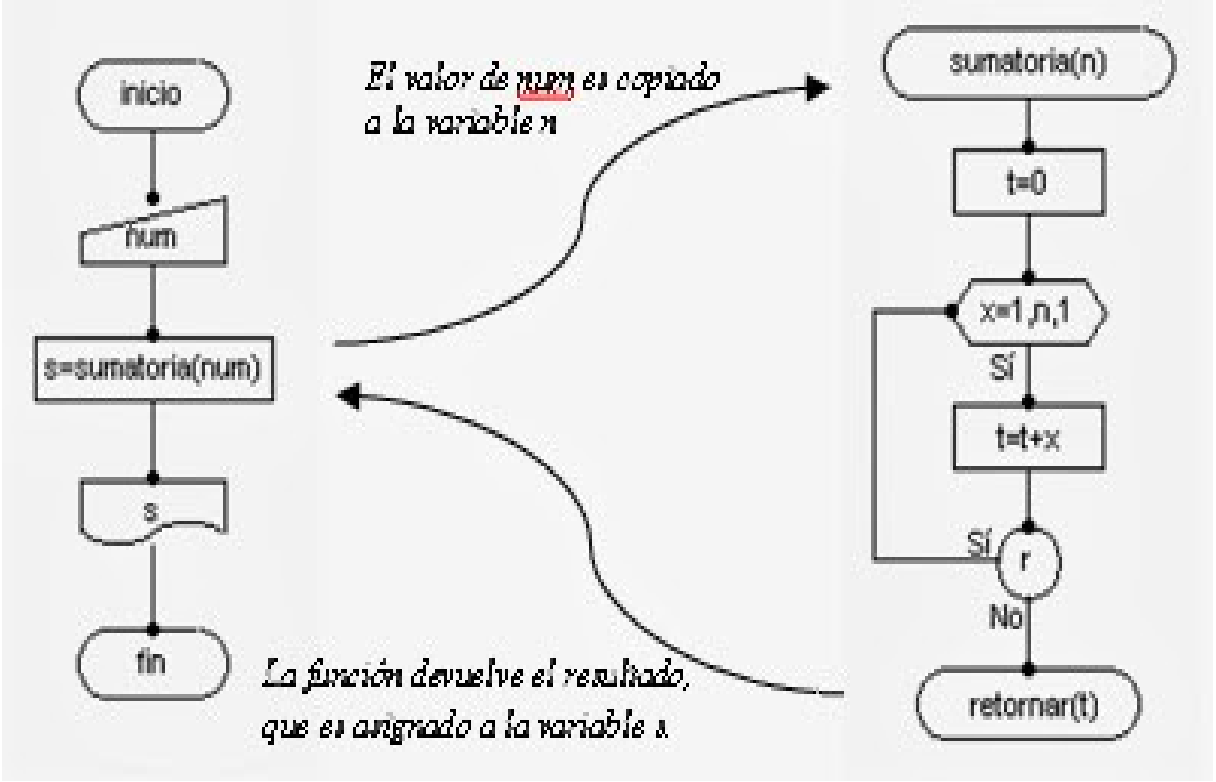
  >>> type(32)

  <type 'int'>
- El **nombre** de la función es *type*. La expresión entre paréntesis recibe el nombre de argumento de la función.
- El **argumento** es un valor o variable que se pasa a la función como parámetro de entrada.
- El **resultado** de la función *type* es el tipo del argumento.
- Es habitual decir que una función “toma” (o recibe) un argumento y “retorna” (o devuelve) un resultado. El resultado se llama valor de retorno.

Programación Modular y llamado a funciones

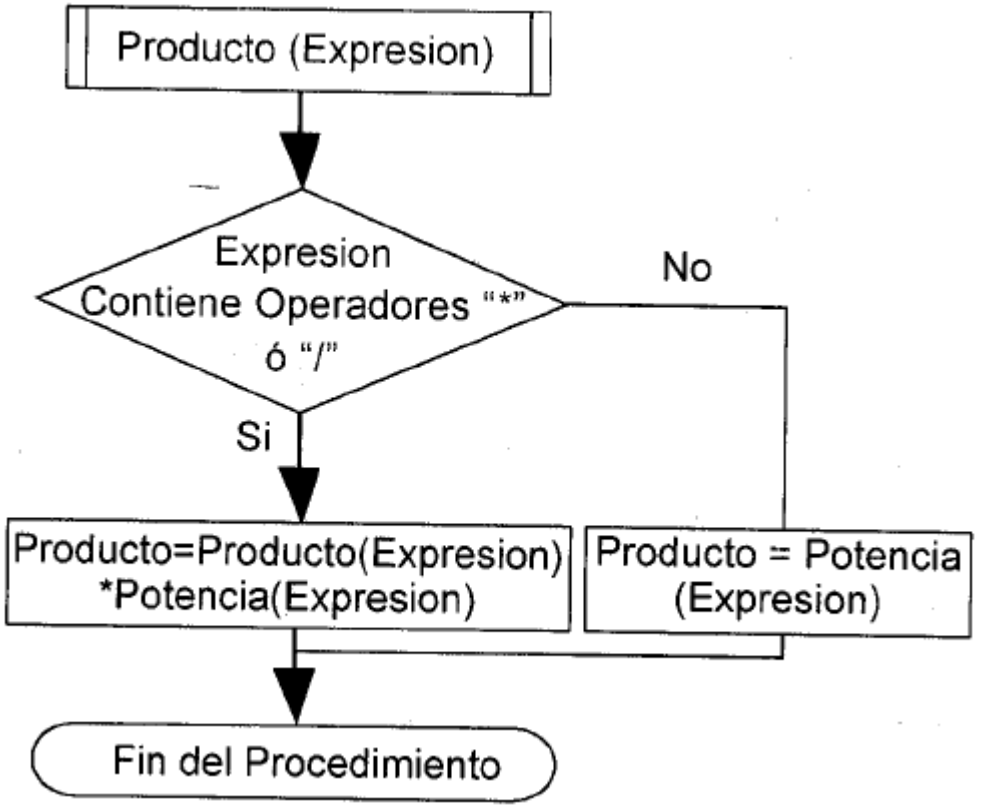


Flujo durante el llamado a la función



- Una llamada a una función es como un desvío en el flujo de la ejecución. En vez de pasar a la siguiente sentencia, el flujo salta al cuerpo de la función, ejecuta todas las sentencias que hay allí, y después vuelve al punto donde lo dejó.

Descripción en diagrama de flujo



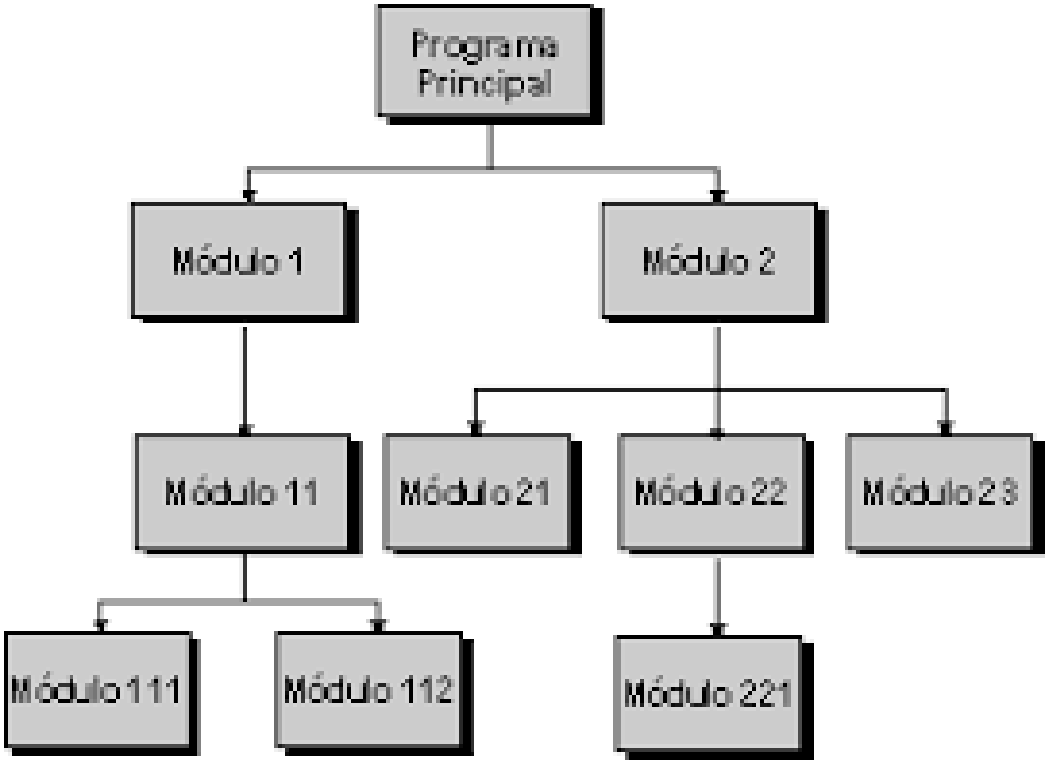
Definición de una función en Python

```
def licencia():
    print("Copyright 2013 Bartolomé Sintes Marco")
    print("Licencia CC-BY-SA 4.0")
    Return


##Aquí inicia el programa principal
print("Este programa no hace nada interesante.")
licencia()
print("Programa terminado.")
```

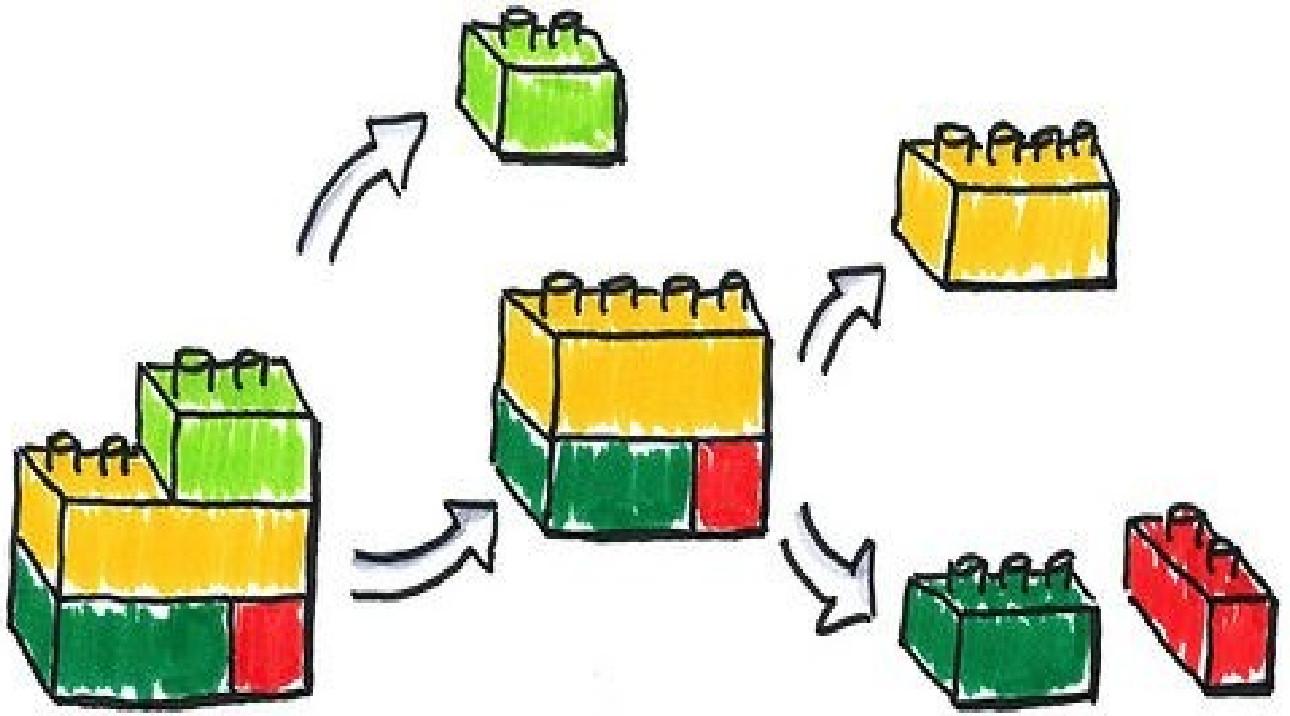
Error al llamar una función antes de haberla definido

```
print("Este programa no hace nada interesante.")
licencia()
print("Programa terminado.")

def licencia():
    print("Copyright 2013 Bartolomé Sintes Marco")
    print("Licencia CC-BY-SA 4.0")
    return
```

Programación modular



Estrategia Divide y Vencerás



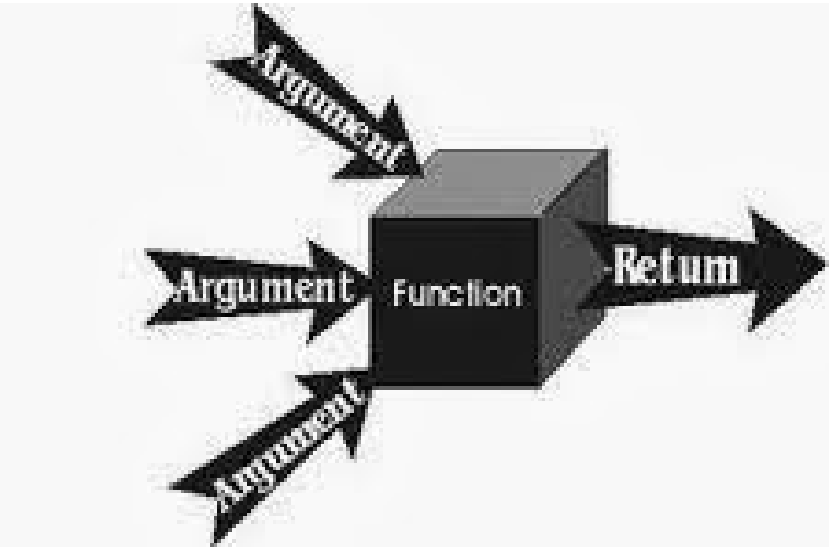
Divide y Vencerás

Utilidad de las funciones

Utilidad de las funciones

- El crear una función nueva te da oportunidad de dar nombre a un grupo de sentencias, lo cual hace a tu programa más fácil de leer, entender, y depurar.
- Las funciones pueden hacer un programa más pequeño, al eliminar código repetido.
- Además, si quieres realizar cualquier cambio en el futuro, sólo tendrás que hacerlo en un único lugar.
- Dividir un programa largo en funciones te permite depurar las partes de una en una y luego ensamblarlas juntas en una sola pieza.
- Las funciones bien diseñadas a menudo resultan útiles para otros muchos programas. Una vez que has escrito y depurado una, puedes reutilizarla.

Función como bloque ó módulo con entradas y salidas



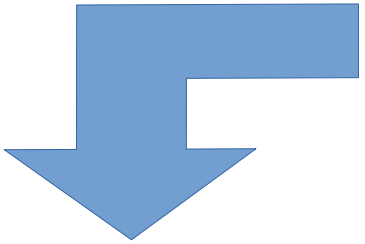
Funciones de una línea

- Si una función es muy simple, puede escribirse de forma abreviada en una línea.

```
def f(x): return 2*x**2+1

n=float(input('ingrese n:'))
r=f(n)
print('resultado:',r)
```

Creación de librerías de funciones

- Una manera de organizar mejor los programas consiste en definir las funciones en archivos aparte del programa principal.



```
def f(x):
    y=2*x**2 + 1
    return y
```

Archivo: funcion.py

```
from función import f
y=f(2)
```

Programa_Principal.py

Nota: El archivo de la librería debe estar en la misma carpeta del programa

Creación librerías con varias funciones

lib_funciones.py

```
from math import*

def f(x): return 2*x**2+1

def g(x): return 3*x**3+5

def esfera(r):
    s=4*pi*r**2
    v=4*pi*r**3/3
    return [s,v]

def cilindro(r,h):
    s=2*pi*r*(r+h)
    v=pi*r**2*h
    return [s,v]
```

Varios argumentos de entrada y salida

Programa Principal

```
from lib_funciones import *
n=float(input('ingrese n:'))
r1=f(n)
print('resultado f:',r1)
r2=g(n)
print('resultado g:',r2)

r=float(input('ingrese radio esfera:'))
[s,v]=esfera(r)
print('resultado area:',s)
print('resultado volumen:',v)

r=float(input('ingrese radio cilindro:'))
h=float(input('ingrese altura cilindro:'))
[s,v]=cilindro(r,h)
print('resultado area:',s)
print('resultado volumen:',v)
```

Ejecución

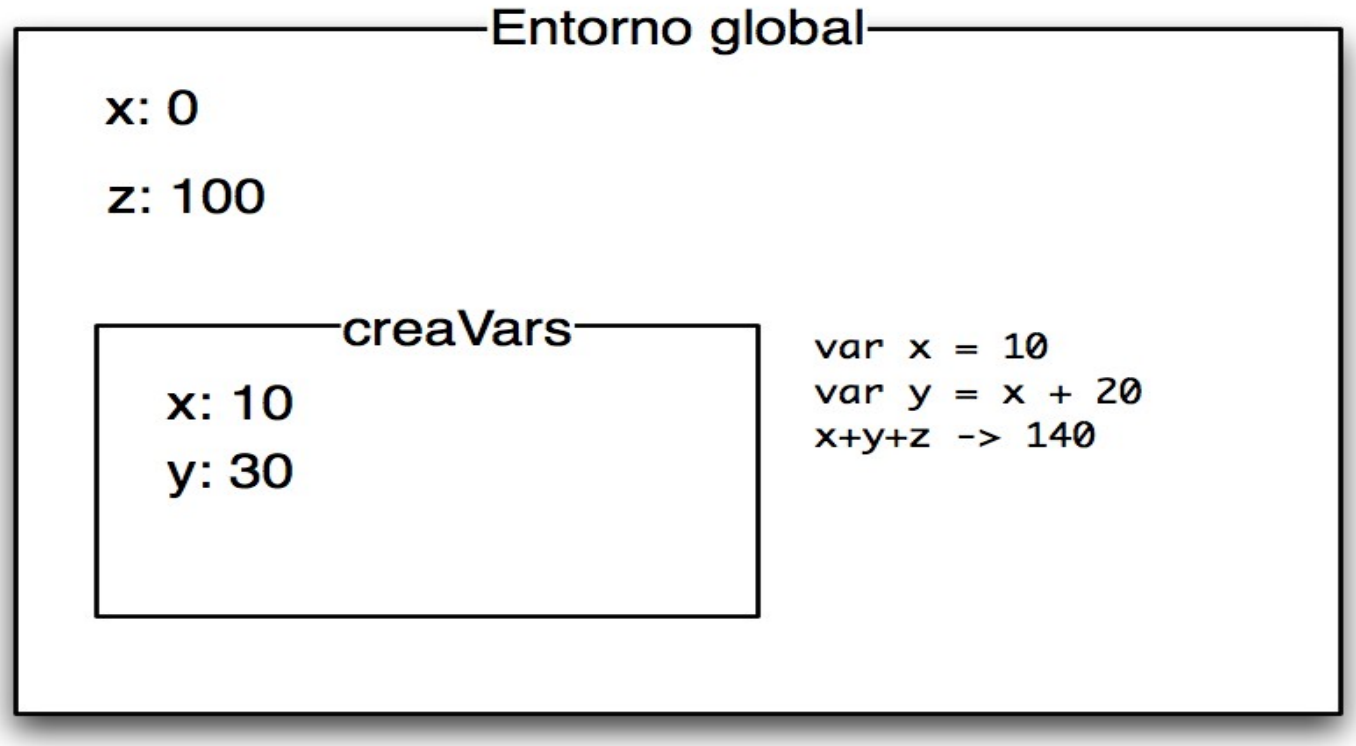
```
ingrese n:2
resultado f: 9.0
resultado g: 29.0

ingrese radio esfera:2
resultado area: 50.26548245743669
resultado volumen: 33.510321638291124

ingrese radio cilindro:2

ingrese altura cilindro:3
resultado area: 62.83185307179586
resultado volumen: 37.69911184307752
```

Variables globales y locales

Python distingue tres tipos de variables: las variables locales y dos tipos de variables libres (globales y no locales):

- variables locales: las que pertenecen al ámbito de la subrutina (y que pueden ser accesibles a niveles inferiores)
- variables globales: las que pertenecen al ámbito del programa principal.
- variables no locales: las que pertenecen a un ámbito superior al de la subrutina, pero que no son globales.

Variables Globales y Locales




```
var x = 0
var z = 100
def creaVars() = {
    ...
}

creaVars()
```

Interpretación de variables globales y locales en Python

- Como Python no es un lenguaje tipificado, el ámbito de pertenencia de la variable debe deducirse del programa siguiendo unas reglas que se detallan más adelante
- Python también permite declarar explícitamente el ámbito en los casos en que se quiera un ámbito distinto al determinado por las reglas.

Declaración explícita de variables globales en Python

- Para identificar explícitamente las variables globales y no locales se utilizan las palabras reservadas **global** y **nonlocal**. Las variables locales no necesitan identificación. La palabra reservada nonlocal se introdujo en Python 3 (PEP 3104).

Ejemplos de variables locales y globales

- Ejemplos interesantes se pueden encontrar en el enlace:

  http://www.mclibre.org/consultar/python/lecciones/python-funciones-1.html

Ejemplo variables locales

```
def subrutina():
    a = 2
    print(a)
    return


a = 5
subrutina()
print(a)
```

Si no se han declarado como globales o no locales, las variables a las que se asigna valor en una función se consideran variables locales, es decir, sólo existen en la propia función, incluso cuando en el programa exista una variable con el mismo nombre

Error al usar una variable de un nivel inferior

```
def subrutina():
    a = 2
    print(a)
    return


subrutina()
print(a)
```

- Las variables locales sólo existen en la propia función y no son accesibles desde niveles superiores

Error al usar una variable local no declarada

```
def subrutina():
    print(a)
    a = 2
    print(a)
    return


a = 5
subrutina()
print(a)
```

- Si en el interior de una función se asigna valor a una variable que no se ha declarado como global o no local, esa variable es local a todos los efectos

Uso de variables globales en funciones inferiores

```
def subrutina():
    print(a)
    return


a = 5
subrutina()
print(a)
```

- Si a una variable no se le asigna valor en una función, Python la considera libre y busca su valor en los niveles superiores de esa función, empezando por el inmediatamente superior y continuando hasta el programa principal.
- Si a la variable se le asigna valor en algún nivel intermedio la variable se considera **no local** y si se le asigna en el programa principal la variable se considera **global**

Ejemplo variable no local

```
def subrutina():
    def sub_subrutina():
        print(a)
        return

    a = 3
    sub_subrutina()
    print(a)
    return

a = 4
subrutina()
print(a)
```

- En el ejemplo siguiente, la variable libre "a" de la función sub_subrutina() se considera no local porque obtiene su valor de una función intermedia

Error variable no definida

```
def subrutina():
    print(a)
    return


subrutina()
print(a)
```

- Si a una variable que Python considera libre (porque no se le asigna valor en la función) tampoco se le asigna valor en niveles superiores, Python dará un mensaje de error

Definición explícita de variables globales

```
def subrutina():
    global a
    print(a)
    a = 1
    return


a = 5
subrutina()
print(a)
```

- Si queremos asignar valor a una variable en una subrutina, pero no queremos que Python la considere local, debemos declararla en la función como global o nonlocal
- En el ejemplo siguiente la variable se declara como global, para que su valor sea el del programa principal

Definición de variable nonlocal

```
def subrutina():
    def sub_subrutina():
        nonlocal a
        print(a)
        a = 1
        return

    a = 3
    sub_subrutina()
    print(a)
    return

a = 4
subrutina()
print(a)
```

- En el ejemplo siguiente la variable se declara como nonlocal, para que su valor sea el de la función intermedia

Error de no asignación de valor a variable superior

```
def subrutina():
    def sub_subrutina():
        nonlocal a
        print(a)
        a = 1
        return

    sub_subrutina()
    print(a)
    return

a = 4
subrutina()
print(a)
```

- Si a una variable declarada global o nonlocal en una función no se le asigna valor en el nivel superior correspondiente, Python dará un error de sintaxis

Bibliografía

http://www.mclibre.org/consultar/python/lecciones/python-funciones-1.html

- Python para informáticos. Explorando la información. Version 2.7.2. Charles Severance