El futuro digital es de todos

MinTIC

Definiendo funciones

Rogerio Orlando Beltrán Castro

Universidad Industrial de Santander

UIS

Mision TIC 2022

Definir funciones en Python

Las funciones se pueden crear en cualquier punto de un programa, escribiendo su definición.

La primera línea de la definición de una función contiene:

- la palabra reservada def
- el nombre de la función (la guía de estilo de Python recomienda escribir todos los caracteres en minúsculas separando las palabras por guiones bajos)
- paréntesis (que pueden incluir los argumentos de la función, como se explica más adelante)

Las instrucciones que forman la función se escriben con sangría con respecto a la primera línea.

Por comodidad, se puede indicar el final de la función con la palabra reservada return (más adelante se explica el uso de esta palabra reservada), aunque no es obligatorio.

Para poder utilizar una función en un programa se tiene que haber definido antes. Por ello, normalmente las definiciones de las funciones se suelen escribir al principio de los programas

Programa que contiene una función y el resultado de la ejecución de ese programa.

```
def licencia():
    print("Copyright Misión TIC'S")
    print("Licencia CC-BY-SA 4.0")
    return


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

Variables en funciones

Como se ha comentado antes, una de las principales ventajas de las subrutinas es que permiten reutilizar el código. Pero copiar y pegar subrutinas de un programa a otro puede producir lo que se llama un conflicto de nombres de variables. En efecto, si la subrutina que pegamos en un programa utiliza alguna variable auxiliar para algún cálculo intermedio y resulta que el programa ya utilizaba una variable con el mismo nombre que esa variable auxiliar, los cambios en la variable que se hagan en la subrutina podrían afectar al resto del programa de forma imprevista.

Para resolver el problema de los conflictos de nombres, los lenguajes de programación limitan lo que se llama el alcance o el ámbito de las variables. Es decir, que los lenguajes de programación permiten que una variable exista únicamente en el interior de una subrutina y no afecte a otras variables de mismo nombre situadas fuera de esa subrutina. Como las subrutinas pueden contener a su vez subrutinas, se suele hablar de niveles: el nivel más alto sería el programa principal, el siguiente nivel serían las subrutinas incluidas en el programa principal y cada vez que hay una subrutina incluida dentro de otra estaríamos bajando un nivel.

Conflictos de nombres de variables

El problema es más complicado de lo que parece a primera vista, porque a menudo también nos interesará que una subrutina pueda modificar variables que estén definidas en otros puntos del programa. Así que los lenguajes de programación tienen que establecer mecanismos para aislar las variables y evitar los conflictos de nombres, pero al mismo tiempo deben permitir el acceso a las variables en los casos que así lo quiera el programador.

Aunque cada lenguaje tiene sus particularidades, el mecanismo más habitual se basa en los siguientes principios:

- cada variable pertenece a un ámbito determinado: al programa principal o a una subrutina.
- las variables son completamente inaccesibles en los ámbitos superiores al ámbito al que pertenecen
- las variables pueden ser accesibles o no en ámbitos inferiores al ámbito al que pertenecen (el lenguaje puede permitir al programador elegir o no esa accesibilidad)
- en cada subrutina, las variables que se utilizan pueden ser entonces:
  - variables locales: las que pertenecen al ámbito de la subrutina (y que pueden ser accesibles a niveles inferiores)
  - variables libres: las que pertenecen a ámbitos superiores pero son accesibles en la subrutina

Conflictos de nombres de variables en Python

Python sigue estos principios generales, pero con algunas particularidades:

En los lenguajes tipificados, como se debe declarar las variables que se utilizan, la declaración se aprovecha para indicar si la variable pertenece a la subrutina o procede de un ámbito superior. Pero 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 (aunque Python también permite declarar explícitamente el ámbito en los casos en que se quiera un ámbito distinto al determinado por las reglas).

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.

Si el programa contiene solamente funciones que no contienen a su vez funciones, todas las variables libres son variables globales. Pero si el programa contiene una función que a su vez contiene una función, las variables libres de esas "subfunciones" pueden ser globales (si pertenecen al programa principal) o no locales (si pertenecen a la función).

Para identificar explícitamente las variables globales y no locales se utilizan las palabras reservadas global y nonlocal. Las variables locales no necesitan identificación.

Variables locales

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, como muestra el siguiente ejemplo:

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


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

Variables locales

Las variables locales sólo existen en la propia función y no son accesibles desde niveles superiores, como puede verse en el siguiente ejemplo:

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


subrutina()
print(a)
```

Variables locales

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. Por ello el siguiente programa da error:

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


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

Variables libres globales o no locales

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, como muestran los siguientes ejemplos:

Variables libres globales o no locales

En el ejemplo siguiente, la variable libre "a" de la función subrutina() se considera global porque obtiene su valor del programa principal:

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


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

Variables libres globales o no locales

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:

```
def subrutina():
    def sub_subrutina():
        print(a)
        return
    a = 3
    sub_subrutina()
    print(a)
    return
a = 4
subrutina()
print(a)
```

Variables libres globales o no locales

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, como muestra el programa siguiente:

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


subrutina()
print(a)
```

Variables declaradas global o nonlocal

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, como muestran los ejemplos siguientes:

Variables declaradas global o nonlocal

En el ejemplo siguiente la variable se declara como global, para que su valor sea el del programa principal:

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


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

Variables declaradas global o nonlocal

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

```
def subrutina():
    def sub_subrutina():
        nonlocal a
        print(a)
        a = 1
        return
    a = 3
    sub_subrutina()
    print(a)
    return
a = 4
subrutina()
print(a)
```

Variables declaradas global o nonlocal

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, como muestra el programa siguiente:

```
def subrutina():
    def sub_subrutina():
        nonlocal a
        print(a)
        a = 1
        return
    sub_subrutina()
    print(a)
    return
a = 4
subrutina()
print(a)
```

Argumentos y devolución de valores

Las funciones en Python admiten argumentos en su llamada y permiten devolver valores. Estas posibilidades permiten crear funciones más útiles y fácilmente reutilizables.

En este apartado se muestran estos conceptos mediante cuatro ejemplos. En ellos, no se pretende encontrar la mejor solución al problema planteado, sino simplemente introducir los conceptos de argumentos y devolución de valores.

Argumentos y devolución de valores

Aunque las funciones en Python pueden acceder a cualquier variable del programa declarándolas como variables globales o no locales, se necesita saber el nombre de las variables, como muestra el ejemplo siguiente:

```
def escribe_media():
    media = (a + b) / 2
    print(f"La media de {a} y {b} es: {media}")
    return


a = 3
b = 5
escribe_media()
print("Programa terminado")
```

El problema de una función de este tipo es que es muy difícil de reutilizar en otros programas o incluso en el mismo programa, ya que sólo es capaz de hacer la media de las variables "a" y "b". Si en el programa no se utilizan esos nombres de variables, la función no funcionaría.

Argumentos y devolución de valores

Para evitar ese problema, las funciones admiten argumentos, es decir, permiten que se les envíen valores con los que trabajar. De esa manera, las funciones se pueden reutilizar más fácilmente, como muestra el ejemplo siguiente:

```
def escribe_media(x, y):
    media = (x + y) / 2
    print(f"La media de {x} y {y} es: {media}")
    return


a = 3
b = 5
escribe_media(a, b)
print("Programa terminado")
```

Pero esta función tiene todavía un inconveniente. Como las variables locales de una función son inaccesibles desde los niveles superiores, el programa principal no puede utilizar la variable "media" calculada por la función y tiene que ser la función la que escriba el valor. Pero eso complica la reutilización de la función, porque aunque es probable que en otro programa nos interese una función que calcule la media, es más difícil que nos interese que escriba el valor nada más calcularlo.

Argumentos y devolución de valores

Para evitar ese problema, las funciones pueden devolver valores, es decir, pueden enviar valores al programa o función de nivel superior. De esa manera, las funciones se pueden reutilizar más fácilmente, como muestra el ejemplo siguiente:

```
def calcula_media(x, y):
    resultado = (x + y) / 2
    return resultado


a = 3
b = 5
media = calcula_media(a, b)
print(f"La media de {a} y {b} es: {media}")
print("Programa terminado")
```

Argumentos y devolución de valores

Para evitar ese problema, las funciones pueden devolver valores, es decir, pueden enviar valores al programa o función de nivel superior. De esa manera, las funciones se pueden reutilizar más fácilmente, como muestra el ejemplo siguiente:

```
def calcula_media(x, y):
    resultado = (x + y) / 2
    return resultado


a = 3
b = 5
media = calcula_media(a, b)
print(f"La media de {a} y {b} es: {media}")
print("Programa terminado")
```

Pero esta función tiene todavía un inconveniente y es que sólo calcula la media de dos valores. Sería más interesante que la función calculara la media de cualquier cantidad de valores.

Argumentos y devolución de valores

Para evitar ese problema, las funciones pueden admitir una cantidad indeterminada de valores, como muestra el ejemplo siguiente:

```
def calcula_media(*args):
    total = 0
    for i in args:
        total += i
    resultado = total / len(args)
    return resultado


a, b, c = 3, 5, 10
media = calcula_media(a, b, c)
print(f"La media de {a}, {b} y {c} es: {media}")
print("Programa terminado")
```

Argumentos y devolución de valores

Las funciones pueden devolver varios valores simultáneamente, como muestra el siguiente ejemplo:

```
def calcula_media_desviacion(*args):
    total = 0
    for i in args:
        total += i
    media = total / len(args)
    total = 0
    for i in args:
        total += (i - media) ** 2
    desviacion = (total / len(args)) ** 0.5
    return media, desviacion
a, b, c, d = 3, 5, 10, 12
media, desviacion_tipica = calcula_media_desviacion(a,
b, c, d)
print(f"Datos: {a} {b} {c} {d}")
print(f"Media: {media}")
print(f"Desviación típica: {desviacion_tipica}")
print("Programa terminado")
```

Conflictos entre nombres de parámetros y nombre de variables globales

En Python no se producen conflictos entre los nombres de los parámetros y los nombres de las variables globales. Es decir, el nombre de un parámetro puede coincidir o no con el de una variable global, pero Python no los confunde: en el ámbito de la función el parámetro hace siempre referencia al dato recibido y no a la variable global y los programas producen el mismo resultado.

Eso nos facilita reutilizar funciones en otros programas sin tener que preocuparnos por este detalle.

Aunque, como siempre, dependiendo de si a la función se le envía como parámetro un objeto mutable o inmutable, la función podrá modificar o no al objeto. En los dos siguientes ejemplos, el parámetro de la función ("b") se llama igual que una de las dos variables del programa principal. En los dos ejemplos se llama dos veces a la función, enviando cada vez una de las dos variables ("a" y "b").

Conflictos entre nombres de parámetros y nombre de variables globales

Ejemplo de conflicto entre nombre de parámetro y nombre de variable global. Objeto mutable

Como en este caso las variables son listas (objetos mutables), la función modifica la lista que se envía como argumento: primero se modifica la lista "a" y a continuación la lista "b". La lista modificada no depende del nombre del parámetro en la función (que es "b"), sino de la variable enviada como argumento ("a" o "b").

```
def cambia(b):
    b += [5]
    return


a, b = [3], [4]
print(f"Al principio       : a = {a} b = {b}")
cambia(a)
print(f"Después de cambia(a): a = {a} b = {b}")
cambia(b)
print(f"Después de cambia(b): a = {a} b = {b}")
print("Programa terminado")
```

Conflictos entre nombres de parámetros y nombre de variables globales

Ejemplo de conflicto entre nombre de parámetro y nombre de variable global. Objeto inmutable

Como en este caso las variables son números enteros (objetos inmutables), la función no puede modificar los números que se envían como argumentos, ni la variable "a" ni la variable "b".

```
def cambia(b):
    b += 1
    return


a, b = 3, 4
print(f"Al principio       : a = {a} b = {b}")
cambia(a)
print(f"Después de cambia(a): a = {a} b = {b}")
cambia(b)
print(f"Después de cambia(b): a = {a} b = {b}")
print("Programa terminado")
```

Paso por valor o paso por referencia

En los lenguajes en los que las variables son "cajas" en las que se guardan valores, cuando se envía una variable como argumento en una llamada a una función suelen existir dos posibilidades:

- paso por valor: se envía simplemente el valor de la variable, en cuyo caso la función no puede modificar la variable, pues la función sólo conoce su valor, pero no la variable que lo almacenaba.
- paso por referencia: se envía la dirección de memoria de la variable, en cuyo caso la función sí que puede modificar la variable.

En Python no se hace ni una cosa ni otra. En Python cuando se envía una variable como argumento en una llamada a una función lo que se envía es la referencia al objeto al que hace referencia la variable. Dependiendo de si el objeto es mutable o inmutable, la función podrá modificar o no el objeto

Paso por valor o paso por referencia

En el ejemplo siguiente, la variable enviada a la función es una variable que hace referencia a un objeto inmutable

```
def aumenta(x):
    print(id(x))
    x += 1
    print(id(x))
    return x


a = 3
print(id(3), id(4))
print(id(a))
print(aumenta(a))
print(a)
print(id(a))
```

Paso por valor o paso por referencia

En el ejemplo siguiente, la variable enviada a la función es una variable que hace referencia a un objeto mutable

```
def aumenta(x):
    print(id(x))
    x += [1]
    print(id(x))
    return x


a = [3]
print(id(a))
print(aumenta(a))
print(a)
print(id(a))
```

Paso por valor o paso por referencia

En el ejemplo siguiente, la variable enviada a la función es una variable que hace referencia a un objeto mutable

```
def aumenta(x):
    print(id(x))
    x += [1]
    print(id(x))
    return x


a = [3]
print(id(a))
print(aumenta(a))
print(a)
print(id(a))
```

Ejercicio 01

Realiza una función que sume dos números pasados por parámetros.

Solución Ejercicio 01

```
def sumaNumeros(numero1, numero2):
    return numero1 + numero2

num1=int(input('Ingrese un numero:'))
num2=int(input('Ingrese un numero:'))

resultado = sumaNumeros(num1, num2)

print(resultado)
```

Ejercicio 02

Realiza una función que indique si un número pasado por parámetro es par o impar.

Solución Ejercicio 02

```
def esPar(numero):
    return (numero % 2 == 0)

numero=int(input('Ingrese un numero:'))

print(esPar(numero))
```

Ejercicio 03

Hacer una función que nos genere un numero aleatorio entre dos parámetros pasados.

Solución Ejercicio 03

```
from random import *

def generaNumeroAleatorio(minimo,maximo):
    try:
        if minimo > maximo:
            aux = minimo
            minimo = maximo
            maximo = aux
        return randint(minimo, maximo)
    except TypeError:
        print("Debes escribir numeros")
        return -1
i=0
numero1=int(input('ingrese un número:'))
numero2=int(input('ingrese un número:'))

while i<5:
    print(generaNumeroAleatorio(numero1,numero2))
    i=i+1
```

Ejercicio 04

Crea una función que calcule el factorial de un número pasado por parámetro

Solución Ejercicio 04

```
def factorial(num):

    resultado = num

    for i in range(num-1,1,-1):
        resultado = resultado * i

    return resultado

numero=int(input('ingrese un número:'))
print(factorial(numero))
```

Ejercicio 05

```
def factorial(num):

    resultado = num

    for i in range(num-1,1,-1):
        resultado = resultado * i

    return resultado

numero=int(input('ingrese un número:'))
print(factorial(numero))
```

Solución Ejercicio 06

Escribir una función que calcule el mínimo, el máximo y la media de tres números.

Ejercicio 06

```
def MaxMinMed(num1,num2,num3):
    if num1==num2 and num2==num3:
        minimo=num1
        maximo=num1
        media=num1
    elif num1>num2 and num1>3:
        maximo=num1
        if num2>num3:
            minimo=num3
        else:
            minimo=num2
    elif num2>num3:
        maximo=num2
        if num1>num3:
            minimo=num3
        else:
            minimo=num1
    else:
        maximo=num3
        if num1>num2:
            minimo=num2
        else:
            minimo=num1
    media=(num1+num2+num3)/3
    return minimo,maximo,media
## Programa Principal
numero1=int(input('ingrese un número:'))
numero2=int(input('ingrese un número:'))
numero3=int(input('ingrese un número:'))
##Llamado a la función
ValorMaximo,ValorMinimo,ValorMedia=MaxMinMed(numero1,numero2,numero3)

print('El valor máximo es:',ValorMaximo,', el valor mínimo es:',ValorMinimo,'la media es:',ValorMedia)
```

Ejercicios

1. Crea una función que reciba 2 números y devuelva el resto de la división del primer número dividido entre el segundo. Imprime el resultado.
2. Crea una función que reciba la base y la altura de un triángulo y devuelva su área. A = ½ bh.
3. Crea una función para calcular el IVA de un producto. Deberá recibir un precio y devolver el precio IVA incluido.
4. Crea una función que reciba un número, calcule su factorial, devuelva el resultado e imprímelo.
5. Escribir una función que tome un carácter y devuelva True si es una vocal, de lo contrario devuelve False.
6. Escribir una funcion sum() y una función multip() que sumen y multipliquen respectivamente todos los números de una lista. Por ejemplo: sum([1,2,3,4]) debería devolver 10 y multip([1,2,3,4]) debería devolver 24.
7. Definir una función inversa() que calcule la inversión de una cadena. Por ejemplo la cadena "estoy probando" debería devolver la cadena "odnaborp yotse"
8. Definir una función es_palindromo() que reconoce palíndromos (es decir, palabras que tienen el mismo aspecto escritas invertidas), ejemplo: es_palindromo ("radar") tendría que devolver True.
9. Definir una función superposicion() que tome dos listas y devuelva True si tienen al menos 1 miembro en común o devuelva False de lo contrario. Escribir la función usando el bucle for anidado.
10. Definir una función generar_n_caracteres() que tome un entero n y devuelva el caracter