Introducción a la programación con Python

Funciones

Funciones

Anteriormente hemos usado funciones nativas que vienen con Python, pero al igual que en otros lenguajes de programación, también podemos definir nuestras propias funciones. Para ello hacemos uso de def.

Cualquier función tendrá un nombre, unos argumentos de entrada, un código a ejecutar y unos parámetros de salida. Al igual que las funciones matemáticas, en programación nos permiten realizar diferentes operaciones con la entrada, para entregar una determinada salida que dependerá del código que escribamos dentro. Por lo tanto, es totalmente análogo al clásico y=f(x) de las matemáticas.

```
def nombre_funcion(argumentos):
    código
    return retorno
```

Funciones

Parten de los siguientes principios:

El principio de reusabilidad: si tenemos un fragmento de código usado en muchos sitios, la mejor solución sería pasarlo a una función. Esto nos evitaría tener código repetido, y que modificarlo fuera más fácil, ya que bastaría con cambiar la función una vez.

El principio de modularidad: En vez de escribir largos trozos de código, es mejor crear módulos o funciones que agrupen ciertos fragmentos de código en funcionalidades específicas, haciendo que el código resultante sea más fácil de leer.

Argumentos

Una función sin parámetros de entrada (argumentos) ni parámetros de salida:

```
def hola_mundo():
    print("hola mundo")
```

De esta forma hemos definido la función “hola_mundo()”. Ahora, si queremos que esta función se ejecute deberíamos llamarla:

```
hola_mundo()
hola mundo
```

Argumentos

Ahora la misma función utilizando un parámetro de entrada:

```
def hola_mundo(nombre):
    print("hola", nombre)
```

Al llamar a la función debemos pasar un valor en el argumento para que se utilice en el código interno de la misma.

```
hola_mundo(2)
hola  2
hola_mundo("Pepe")
hola  Pepe
```

Return

La sentencia return realiza 2 cosas:

- Salir de la función y transferir la ejecución de vuelta a donde se realizó la llamada.
- Devolver uno o varios parámetros, fruto de la ejecución de la función.

```
def hola_mundo(nombre):
    print("hola", nombre)
    return
    print("adios", nombre)
```

```
hola_mundo("Fernando")
hola Fernando
```

```
def suma(a, b):
    resultado = a+b
    return resultado


numero1=10
numero2=14
miSuma = suma(numero1, numero2)
print(miSuma)
```

Return

Se pueden devolver más de 1 argumento de salida:

```
def sumaYresta(a, b):
    suma = a+b
    resta = a-b
    return suma, resta


numero1=10
numero2=14
miSuma, miResta = sumaYresta(numero1, numero2)
print(miSuma)
print(miResta)
```

Scope o ámbito de las variables

Es posible que al utilizar una función en un programa principal, estos contengan variables con el mismo nombre. Esto podría llevar a conflictos y que los programas no funcionen como se espera de ellos.

Para esto, en python se utilizan 3 tipos de variable:

- Variables locales
- Variables globales
- Variables no locales

Scope o ámbito de las variables

Las variables locales pertenecen al ámbito de la función.

Solo existen en la propia función y no son accesibles a niveles superiores.

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.

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

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

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

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

Scope o ámbito de las variables

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.

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

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

Scope o ámbito de las variables

Si queremos asignar valor a una variable en una función, pero no queremos que Python la considere local, debemos declararla en la función como global o nonlocal.

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

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

Ejemplo 1

¿Qué resultado dará este programa?

```
def subrutina():
    global a
    print(a)
    a += 10
    return

a = 33
subrutina()
print(a)
```

Ejemplo 2

¿Qué resultado dará este programa?

```
def subrutina():
    global a
    print(a)
    a = 21
    return

subrutina()
a = 20
print(a)
```

Ejemplo 3

¿Qué resultado dará este programa?

```
def subrutina():
    global a
    a = 10
    print(a)
    return

a = 33
subrutina()
print(a)
```

Ejemplo 4

¿Qué resultado dará este programa?

```
def subrutina():
    nonlocal a
    print(a)
    a = 32
    return

a = 31
subrutina()
print(a)
```

Ejemplo 5

¿Qué resultado dará este programa?

```
def subrutina():
    print(a)
    a = 11
    return

a = 10
subrutina()
print(a)
```

Ejemplo 6

¿Qué resultado dará este programa?

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

a = 4
b = 3
subrutina()
print(a)
```

Ejemplo 7

¿Qué resultado dará este programa?

```
def subrutina_1():
    a = 20
    print(a)
    return

def subrutina_2():
    global a
    a = 30
    print(a)
    return

a = 10
subrutina_1()
print(a)
subrutina_2()
print(a)
```

Ejemplo 8

¿Qué resultado dará este programa?

```
def subrutina():
    def sub_subrutina():
        a = 5
        print(a)
        return

    a = 4
    sub_subrutina()
    print(a)
    return

a = 3
subrutina()
print(a)
```

Ejemplo 9

¿Qué resultado dará este programa?

```
def subrutina():
    def sub_subrutina():
        a = 3
        print(a)
        a = 5
        return

    a = 3
    sub_subrutina()
    print(a)
    return

a = 4
sub_subrutina()
print(a)
```

Ejemplo 10

¿Qué resultado dará este programa?

```
def subrutina():
    def sub_subrutina():
        a = 5
        print(a)
        return

    global a
    a = 4
    sub_subrutina()
    print(a)
    return

a = 3
subrutina()
print(a)
```

Ejemplo 11

¿Qué resultado dará este programa?

```
def subrutina():
    def sub_subrutina():
        global a
        a = 5
        print(a)
        return

    global a
    a = 4
    sub_subrutina()
    print(a)
    return

a = 3
subrutina()
print(a)
```

Ejemplo 12

¿Qué resultado dará este programa?

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

    a += 3
    sub_subrutina()
    print(a)
    return

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

Ejemplo 13

¿Qué resultado dará este programa?

```
def subrutina():
    def sub_subrutina():
        global a
        a = 5
        print(a)
        return

    a = 4
    sub_subrutina()
    print(a)
    return

a = 3
subrutina()
print(a)
```