

## Guía Teórica N°01

### Profundizando en Funciones

Ha llegado el momento de seguir aprendiendo mucho más sobre unos de los lenguajes de programación con mayor reconocimiento en la comunidad de programadores y desarrolladores, Python.

En el curso básico aprendimos sobre todos aquellos términos teóricos necesarios y colocamos en práctica esas bases teóricas que es donde está el punto clave de la programación.

Ahora, es turno de continuar y para ello, en este Bloque #01 repasaremos y profundizaremos un poco más sobre uno de los temas más importantes a considerar cuando programamos, las funciones. Así que haremos un breve repaso sobre qué son las funciones, cómo se define y luego, entraremos en contenido nuevo sobre este tema tan importante.

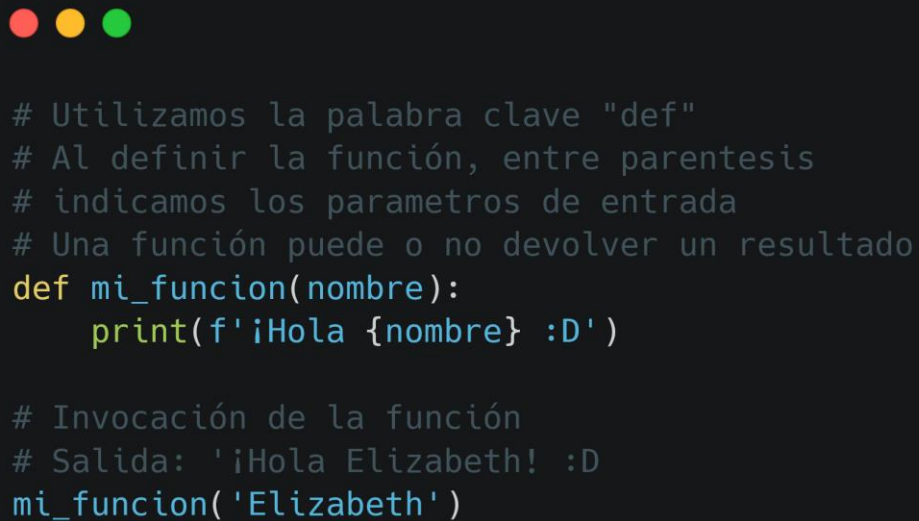
#### Breve Repaso:

Una función es un bloque de código con un nombre asociado, que recibe cero o más argumentos como entrada, sigue una secuencia de sentencias, la cuales ejecuta una operación deseada y devuelve un valor y/o realiza una tarea, este bloque puede ser llamado cuando se necesite.

El uso de funciones es un componente muy importante del paradigma de la programación llamada estructurada, y tiene varias ventajas:

- **Modularización:** Permite segmentar un programa complejo en una serie de partes o módulos más simples, facilitando así la programación y el depurado.
- **Reutilización:** Permite reutilizar una misma función en distintos programas.

Recordemos que la sintaxis básica de una función en Python es de la siguiente manera:

A dark-themed terminal window with three colored window control buttons (red, yellow, green) in the top-left corner. It contains Python code for defining and calling a function. The code is as follows:

```
# Utilizamos la palabra clave "def"
# Al definir la función, entre parentesis
# indicamos los parametros de entrada
# Una función puede o no devolver un resultado
def mi_funcion(nombre):
    print(f'¡Hola {nombre} :D')

# Invocación de la función
# Salida: '¡Hola Elizabeth! :D'
mi_funcion('Elizabeth')
```

Habiendo refrescado un poco sobre qué son las funciones, cuál es su objetivo y cómo utilizarlas, ahora es turno de mencionar algunos detalles nuevos al momento de hacer uso de estos fragmentos de código con una tarea en especial que nosotros definamos.

### **Ámbito de variables:**

El ámbito de una variable es el contexto en el que existe esa variable. Así, una variable existe en dicho ámbito a partir del momento en que se crea y deja de existir cuando desaparece su ámbito.

Además, las variables son accesibles solo desde su propio ámbito, pero con alguna excepción, como veremos más adelante.

Los principales tipos de ámbitos en Python son dos:

- **Ámbito local:** corresponde con el ámbito de una función, que existe desde que se invoca a una función hasta que termina su ejecución. En un programa, el ámbito local corresponde con las líneas de código de una función. Dicho ámbito se crea cada vez que se invoca a la función. Cada función tiene su ámbito local. No se puede acceder a las variables de una función desde fuera de esa función o desde otra función.
- **Ámbito global:** corresponde con el ámbito que existe desde el comienzo de la ejecución de un programa. Todas las variables definidas fuera de cualquier función corresponden al ámbito global, que es accesible desde cualquier punto del programa, incluidas las funciones.

### Variables Locales:

En Python las variables locales son aquellas definidas dentro de una función. Solamente son accesibles desde la propia función y dejan de existir cuando esta termina su ejecución. Los parámetros de una función también son considerados como variables locales.

```
def saludar():
    # esta variable local pertenece
    # al ámbito local de la función saludar
    saludo = '¡Hola!'
    print(saludo)

def sumar(sumando1, sumando2):
    # sumando1, sumando2 y suma
    # son variables locales que corresponden al ámbito local
    # de la función sumar
    suma = sumando1 + sumando2

    return suma

saludar()
print(sumar(4, 5))
```

## Variables Globales:

Las variables globales en Python son aquellas definidas en el cuerpo principal del programa fuera de cualquier función. Son accesibles desde cualquier punto del programa, incluso desde dentro de funciones (¡**OJO!**, podemos acceder al valor de esta pero no modificarla). También se puede acceder a las variables globales de otros programas o módulos importados.

```
#variable global definida en el cuerpo principal
saludo = '¡Hola, Mundo!'

def saludar():
    print(saludo)

    saludo = 'Bienvenidos a una nueva clase'
    # En la línea anterior
    # Probablemente nos suceda un error

saludar()
```

Si queremos modificar el valor de una variable que se encuentra definida de manera global dentro de una función, debemos indicar la palabra clave **global** seguidamente del nombre de la variable global que deseamos modificar.

```
#variable global definida en el cuerpo principal
saludo = '¡Hola, Mundo!'

def saludar():
    global saludo

    print(saludo)

    saludo = 'Bienvenidos a una nueva clase'
    # Ahora si podremos modificar el valor
    # De la variable global 'saludo'

saludar()
```

## Pase de parámetros por valor y por referencia:

Dependiendo del tipo de dato que enviemos a la función, podemos diferenciar dos comportamientos:

- **Pase por valor:** Se crea una copia local de la variable dentro de la función.
- **Pase por referencia:** Se maneja directamente la variable, los cambios realizados dentro de la función le afectarán también fuera.

Tradicionalmente, **los tipos simples se pasan por valor** (Enteros, flotantes, cadenas de caracteres y booleanos) y **los tipos compuestos se pasan por referencia** (Listas, diccionarios, conjuntos, entre otros).

### Ejemplo de pase por valor:

```
"""
Los números se pasan por valor y crean una copia
dentro de la función, por eso no les afecta
externamente lo que hagamos con ellos:
"""

def doblar_valor(numero):
    numero *= 2

n = 10
doblar_valor(n)
print(n)
```

### Ejemplo de pase por referencia:

```
"""
Sin embargo las listas u otras colecciones,
al ser tipos compuestos se pasan por referencia,
y si las modificamos dentro de la función estaremos
modificándolas también fuera:
"""

def doblar_valores(numeros):
    for i,n in enumerate(numeros):
        numeros[i] *= 2

ns = [10,50,100]
doblar_valores(ns)
print(ns)
```

### Pase de Argumentos Indeterminados:

Quizá en alguna ocasión no sabemos de antemano cuantos elementos vamos a enviar a una función. En estos casos podemos utilizar los parámetros indeterminados por posición y por nombre.

- **Por posición:** Para recibir un número indeterminado de parámetros por posición, debemos crear una lista dinámica de argumentos (una tupla en realidad) definiendo el parámetro con un asterisco.
- **Por nombre:** Para recibir un número indeterminado de parámetros por nombre (clave-valor o en inglés *keyword args*), debemos crear un diccionario dinámico de argumentos definiendo el parámetro con dos asteriscos.

```

def indeterminados_posicion(*args):
    for arg in args:
        print(arg)

def indeterminados_nombre(**kwargs):
    print(kwargs)

indeterminados_posicion(5, "Hola", [1, 2, 3, 4, 5])
indeterminados_nombre(n=5, c="Hola", l=[1, 2, 3, 4, 5])

```

## Funciones Recursivas:

Se dice que una función es recursiva cuando el cuerpo de la función utiliza a la propia función. Dentro de una función recursiva suelen distinguirse dos partes:

- **Caso Base:** Son aquellos que para su solución no requieren utilizar la función que se está definiendo.
- **Caso Recursivo:** Son aquellos que sí que requieren utilizar la función que se está definiendo.

Las definiciones recursivas funcionan siempre y cuando las llamadas recursivas se realicen de forma que en algún momento se lleguen a los casos base.

**NOTA:** Funcionan de forma similar a las iteraciones, pero debemos encargarnos de planificar el momento en que dejan de llamarse a sí mismas o tendremos una función recursiva infinita.

```

def factorial(n):
    # Caso Base
    if n == 0:
        return 1
    else:
        # Caso Recursivo
        return n * factorial( n-1 )

```

## **¡Nos vemos en la próxima!**

Con esto, hemos aprendido mucho más sobre las funciones y podemos comprender muy bien el comportamiento de estas al momento de trabajar en nuestros códigos. Es importante conocer muy bien todo lo relacionado a las funciones porque en este curso nos centraremos en el trabajo con módulos y paquetes los cuales tendremos la posibilidad de compartir con otros programadores y es vital conocer todo tipo de situación que se nos presente para evitar errores a futuros.

Ahora continuaremos profundizando en la Programación Orientada a Objetos en el próximo bloque 😊