

# **Programación 1**

## **1º Año**

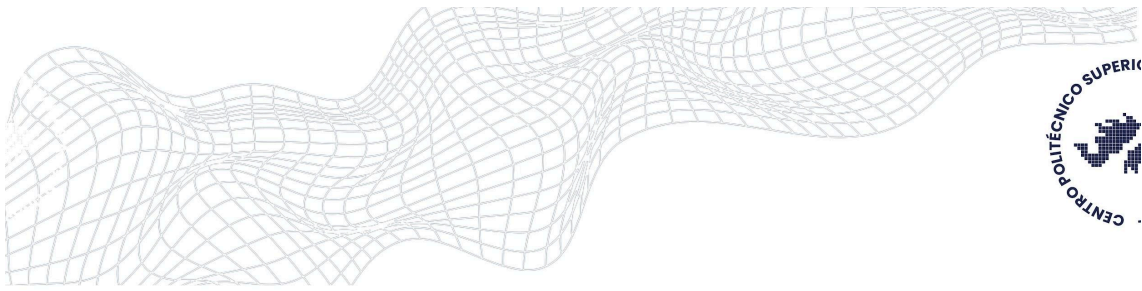
### **Clase N° 6**

#### **Funciones y procedimientos**

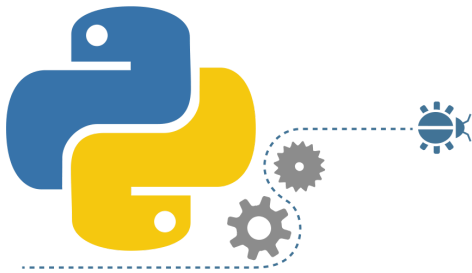
##### **Contenido**

En la clase de hoy trabajaremos los siguientes temas:

- Introducción a los subalgoritmos o subprogramas.
- Funciones, Declaración de funciones, Invocación a las funciones.
- Procedimientos (subrutinas).
- Sustitución de argumentos/parámetros.
- Ámbito: variables locales y globales.
- Recursión (recursividad).



## 1. Presentación



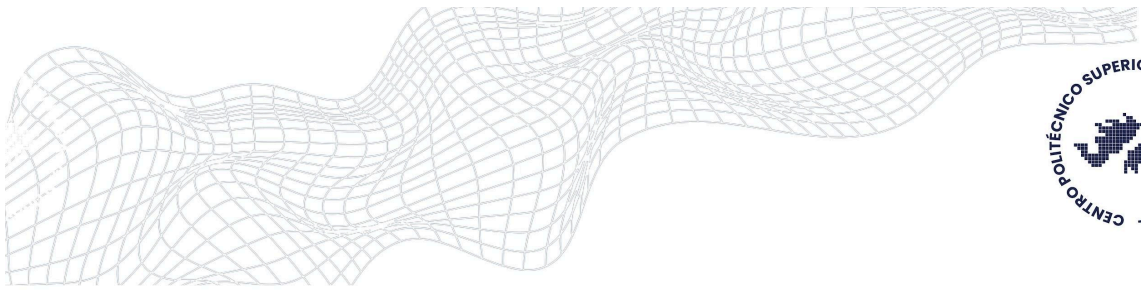
¡Bienvenidos y bienvenidas a una nueva clase de Programación en Python!

Hoy nos adentramos en un tema fundamental para construir programas ordenados, reutilizables y más eficientes:

### **Las funciones, los procedimientos y la recursividad...**

Vamos a descubrir cómo dividir un programa en bloques lógicos que realizan tareas específicas, cómo reutilizar código, pasar información entre partes del programa y cómo una función ¡puede llamarse a sí misma!

Te aseguramos que después de esta clase vas a poder escribir programas más claros, más organizados y mucho más potentes. 🌟



## Pregunta para reflexionar



Imagina que estás por hacer un gran viaje.

Llevás una lista con muchas tareas: preparar la valija, comprar pasajes, cargar el celular, etc.

¿No sería más práctico agrupar esas tareas en pasos o rutinas que puedas repetir o ajustar según el destino?


📌 ¿Cómo podrías organizar esas tareas para que no se repitan y siempre funcionen, sin importar el viaje?

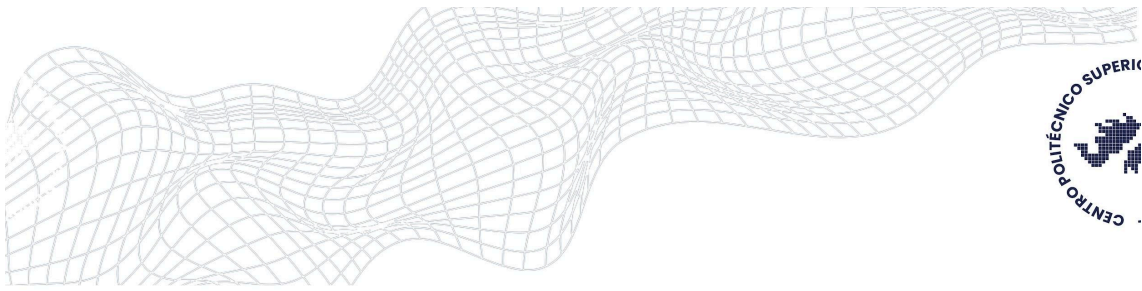


*(Tomémonos unos minutos para pensar)*



Lo que acabas de imaginar es, en esencia, modularidad.

Hoy vamos a aprender cómo aplicarla en programación usando funciones y procedimientos. También veremos un concepto poderoso: la recursión .



## Vídeo Tutorial



En el siguiente video se presenta una introducción clara y visual sobre un primer vistazo a utilizar funciones en Python.

Además, muestra ejemplos simples para comprender cómo funcionan.

### [Introducción a las funciones en Python](#)

 **Te proponemos mirar el video y luego responder:**

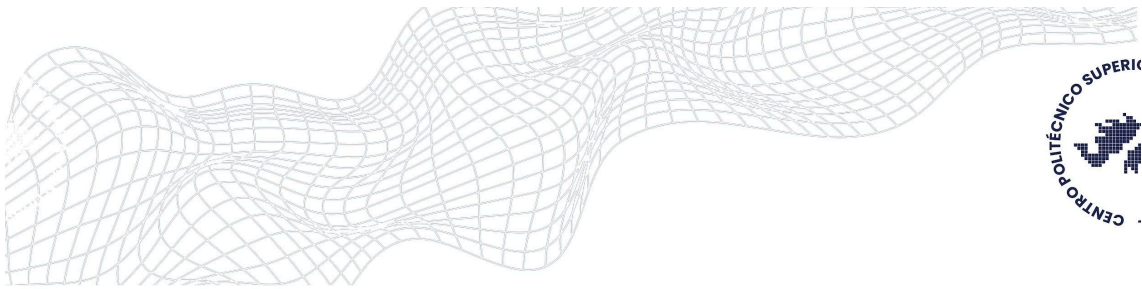


*Preguntas para reflexionar después del video*

→ ¿En qué se parece una función en Python a una función matemática como  $f(x)=x+2$ ?

Pensá en cómo se usa la entrada, qué devuelve y cómo se escribe.

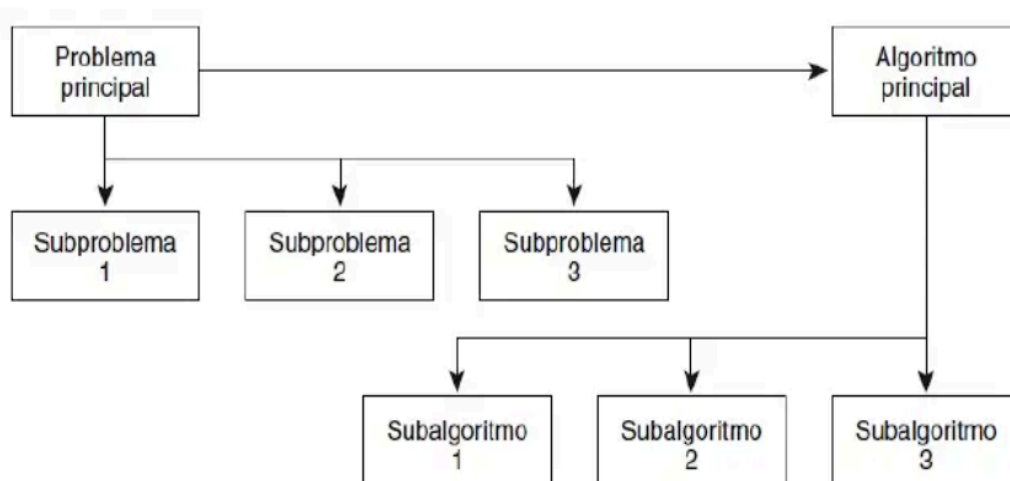
- ¿Por qué creés que usar funciones ayuda a que un programa sea más claro o más fácil de modificar?
- ¿Te imaginás algún caso de la vida cotidiana donde sería útil tener una función programada? ¿Qué tarea podrías automatizar?

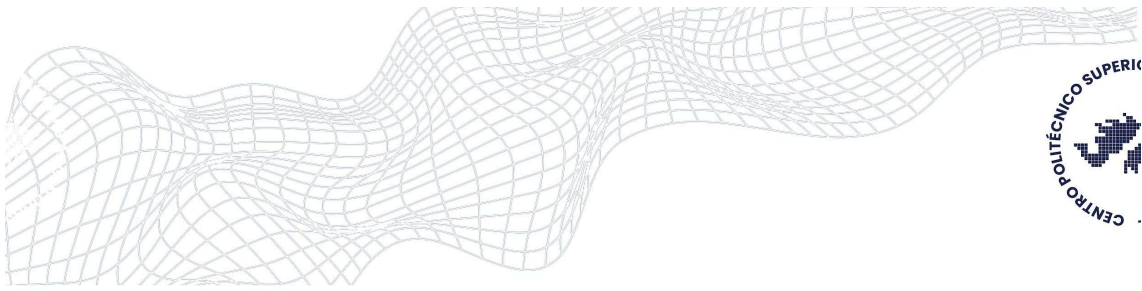


## 2. Desarrollo

### 2.1 Introducción a los subalgoritmos o subprogramas

Los subalgoritmos o subprogramas son bloques de código que realizan una tarea específica y se pueden reutilizar en diferentes partes de un programa. Permiten dividir un programa en partes más pequeñas y manejables, lo que facilita el desarrollo, la depuración y el mantenimiento del código. Los subalgoritmos pueden ser funciones o procedimientos.





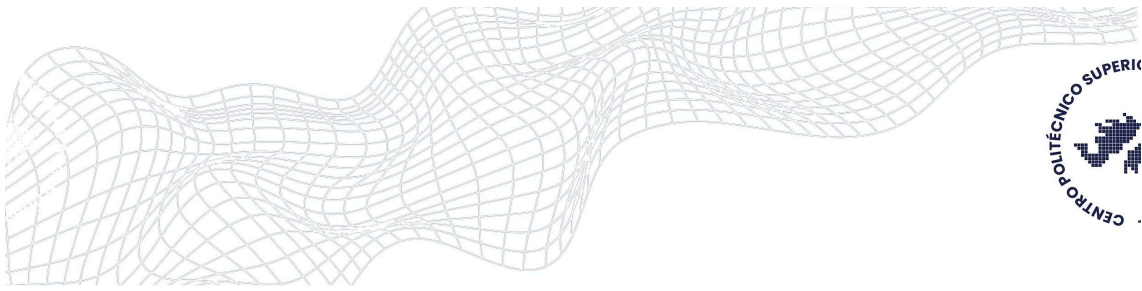
## 2.2 Funciones

Las funciones son bloques de código que reciben cero o más argumentos, ejecutan instrucciones y devuelven un resultado. En Python, se definen con la palabra clave `def`, seguida del nombre, paréntesis (con o sin parámetros) y dos puntos. El cuerpo de la función debe ir indentado.

```
def saludar(nombre):  
    """  
    Esta función recibe un nombre como parámetro y muestra un  
    saludo.  
    """  
    print("¡Hola, " + nombre + "! Bienvenido.")  
  
# Ejemplo de uso  
saludar("Juan")
```

En este ejemplo, **saludar** recibe un parámetro (**nombre**) y muestra un mensaje personalizado. Al llamarla con "**Juan**", se imprime: **¡Hola, Juan! Bienvenido.**

Las funciones permiten organizar el código, evitar repeticiones y facilitar la lectura y el mantenimiento.



## Recordatorio



Recordar que la documentación entre comillas triples (""" ... """) después de la definición de la función se conoce como el docstring de la función y proporciona una descripción de lo que hace la función.

## Vídeo Tutorial

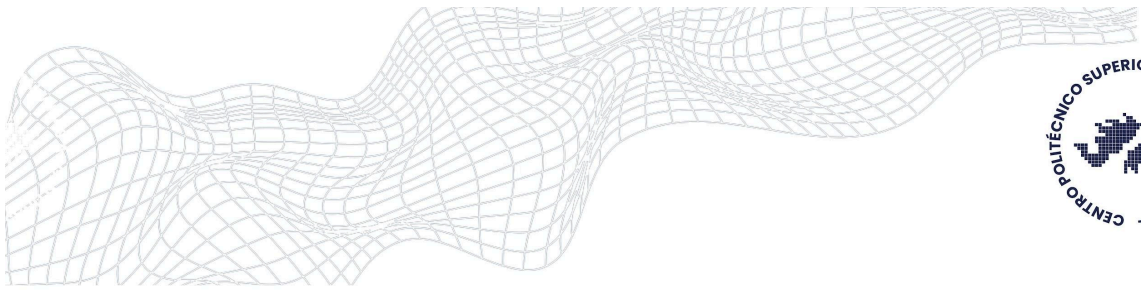


En el siguiente video se presenta una introducción clara y visual sobre cómo crear y utilizar funciones en Python.

Además, muestra ejemplos simples para comprender cómo pasan y retornan valores.

## [Funciones personalizadas en Python](#)





## 2.2.1 Declaración de funciones

La declaración de una función implica definir su nombre, los argumentos que recibe y el tipo de valor que devuelve. En Python, no es necesario especificar el tipo de valor de retorno, ya que el lenguaje es de tipado dinámico. Sin embargo, puedes agregar comentarios en el código utilizando la sintaxis de las anotaciones de tipo para indicar el tipo esperado de los argumentos y del valor de retorno de la función.

Aquí hay un ejemplo:

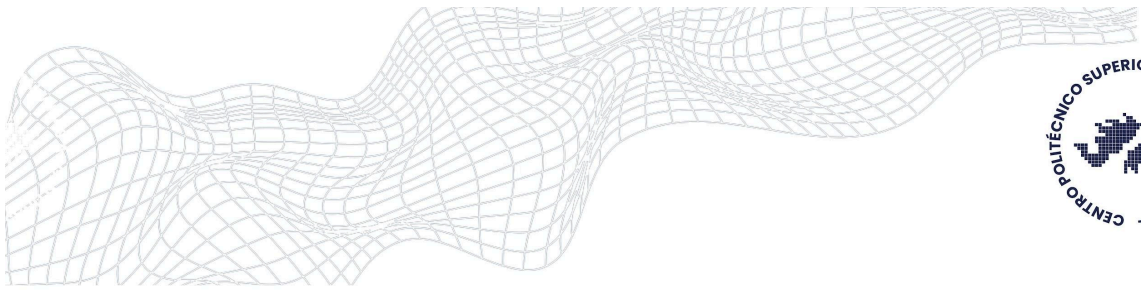
```
def saludar(nombre: str) -> str:  
    return "¡Hola, " + nombre + "!"
```

- def es la palabra clave utilizada para definir una función en Python.
- saludar es el nombre de la función.
- (nombre: str) es el parámetro de la función. En este caso, se espera que se proporcione un argumento de tipo str y se asignará a la variable nombre.
- -> str indica el tipo de valor que la función devuelve. En este caso, la función devuelve un valor de tipo str.

```
return "¡Hola, " + nombre + "!"
```

return es una declaración utilizada para devolver un valor de una función. En este caso, se devuelve una cadena de texto que contiene





el saludo formateado.

"¡Hola, " es una cadena de texto que representa la primera parte del saludo.

+ nombre + es la concatenación del valor de la variable nombre (el argumento proporcionado cuando se llama a la función) con la cadena de texto.

!" es una cadena de texto que representa la última parte del saludo.

En resumen, la función saludar toma un nombre como argumento, lo concatena con una cadena de texto que representa un saludo y devuelve el resultado como una cadena de texto.

Es importante destacar que, en este caso, el tipo de los parámetros y el tipo de retorno están anotados utilizando la sintaxis de anotaciones de tipo de Python. Esto no afecta directamente el comportamiento de la función, pero puede ser útil para documentar y comprender mejor los tipos esperados y retornados por una función.

### 2.2.2 Declaración de devolución

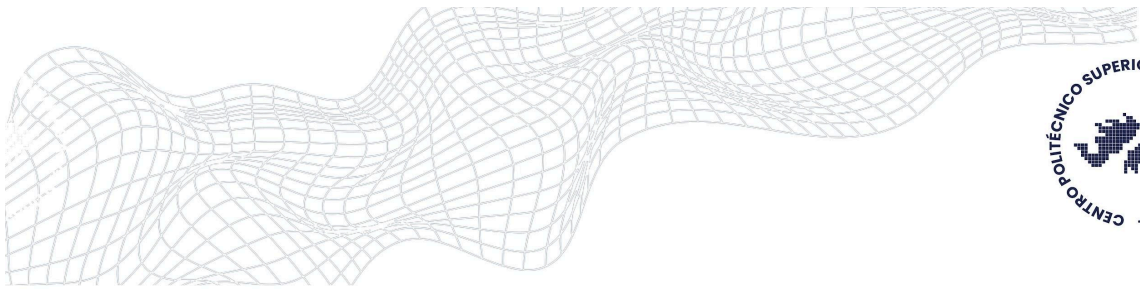
una función puede usar la return declaración para devolver un valor como resultado de la función. Este valor puede asignarse a una variable o usarse en otras expresiones. Por ejemplo:

```
def add_numbers(a, b):  
    return a + b  
  
result = add_numbers(3, 5)  
print(result) # Output: 8
```

### 2.2.3 Argumentos de palabras clave

al llamar a una función, puede especificar argumentos por sus nombres de parámetro. Esto le permite pasar argumentos en cualquier orden y hace que el código sea más legible.

Por ejemplo:



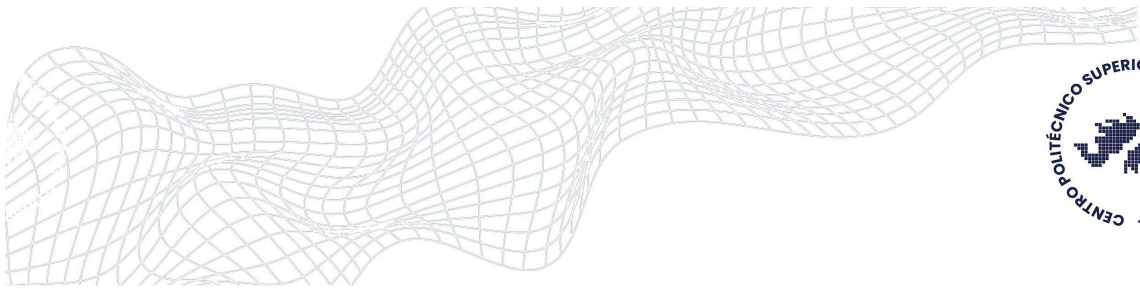
```
def describe_person(name, age):  
    print("Name:", name)  
    print("Age:", age)  
  
describe_person(age=25, name="John")  
# Output:  
# Name: John  
# Age: 25
```

#### 2.2.4 Número variable de argumentos

las funciones de Python pueden recibir un número variable de argumentos utilizando la sintaxis `*args` o `**kwargs`. `*args` permite pasar múltiples argumentos posicionales, mientras que `**kwargs` permite pasar múltiples argumentos de palabras clave. Aquí hay un ejemplo:

```
def calculate_total(*args):  
    total = sum(args)  
    return total  
  
result = calculate_total(2, 4, 6, 8)  
print(result) # Output: 20
```

Estos son solo algunos aspectos adicionales de las funciones en Python. Las funciones son un concepto clave en la programación, ya que ayudan a organizar el código, promueven la reutilización y facilitan la comprensión y el mantenimiento de sus programas.



### 2.2.5 Invocación a las funciones

La invocación o llamada a una función se realiza escribiendo el nombre de la función seguido de paréntesis que pueden contener los argumentos necesarios. Aquí hay un ejemplo de invocación a la función "saludar" del ejemplo anterior:

```
nombre = "Juan"
mensaje = saludar(nombre)
print(mensaje) # Imprimirá "¡Hola, Juan!"
```

En este caso, se pasa el valor de la variable "nombre" como argumento a la función "saludar" y se guarda el valor de retorno en la variable "mensaje".

### 2.3 Procedimientos (subrutinas)

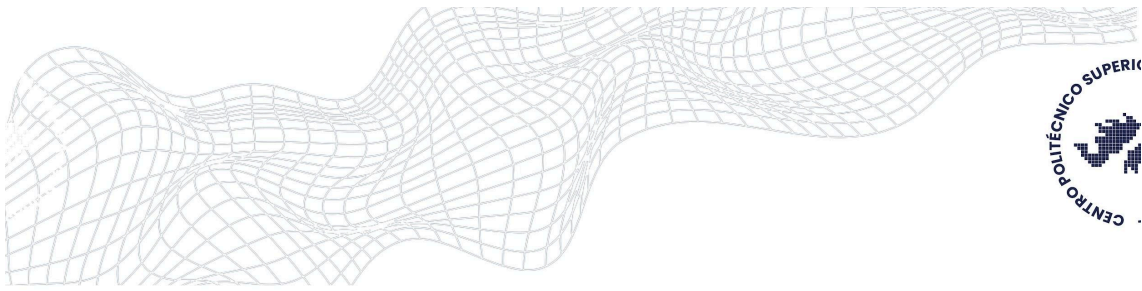
Los procedimientos, también conocidos como subrutinas, son subalgoritmos similares a las funciones, pero no devuelven un valor. En Python, se definen de la misma manera que las funciones, pero no se utiliza la instrucción "return". Aquí hay un ejemplo de un procedimiento que imprime un mensaje:

```
def imprimir_mensaje():
    print("Este es un mensaje")
```

Para invocar a un procedimiento, simplemente se escribe su nombre seguido de paréntesis, sin necesidad de asignar el valor de retorno a una variable. Por ejemplo:

```
imprimir_mensaje()
```

El procedimiento "imprimir\_mensaje" se ejecutará y mostrará el mensaje en la consola.



### Ejercicio de reflexión

***Si un procedimiento no devuelve nada, ¿para qué sirve?*** Pensá en acciones como mostrar info, guardar algo en un archivo o registrar eventos.



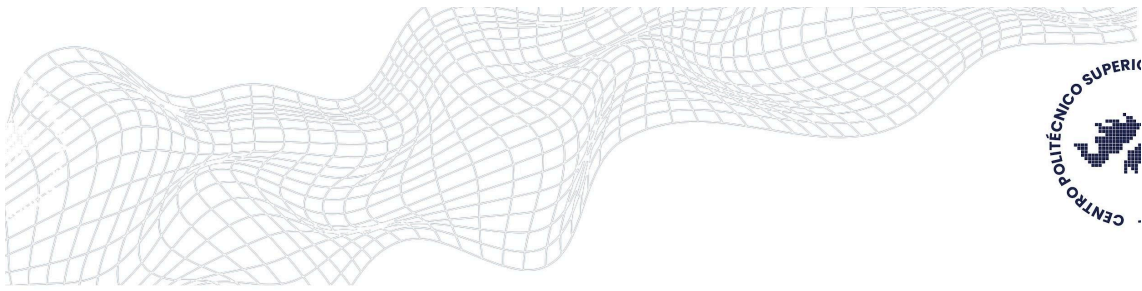
**Abrí Visual Studio Code y creá un procedimiento que haga algo más interesante que simplemente imprimir un mensaje.**

Podés simular un inicio de sesión, registrar información en consola o mostrar una secuencia de pasos. Elegí la acción que prefieras y ejecutalo para ver cómo funciona.

## 2.4 Sustitución de argumentos/parámetros

La sustitución de argumentos o parámetros en Python se refiere al proceso de proporcionar valores concretos a los parámetros de una función al llamarla. Estos valores se utilizan para realizar cálculos o ejecutar acciones dentro de la función.

Cuando se define una función, se pueden especificar uno o más parámetros que actúan como espacios reservados para los valores que se utilizarán cuando se llame a la función. Estos parámetros pueden ser variables que almacenan los valores pasados a la función.



Aquí hay un ejemplo sencillo para ilustrar la sustitución de argumentos en Python:

```
def saludar(nombre):  
    print("¡Hola, " + nombre + "!")  
  
# Llamada a la función con un argumento concreto  
saludar("Juan")
```

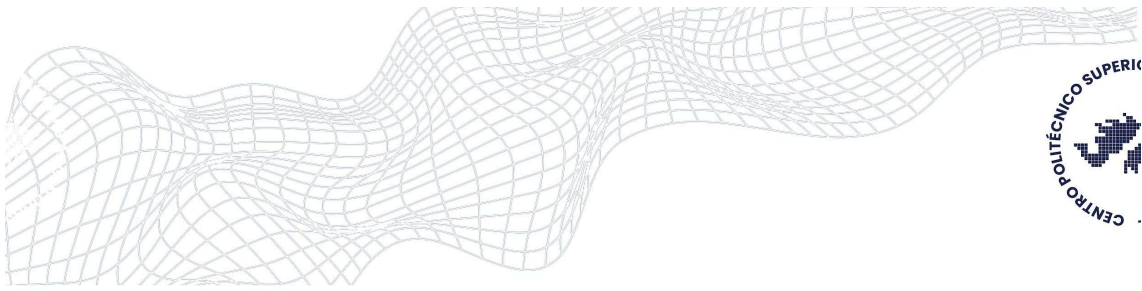
En este caso, se define una función llamada saludar que tiene un parámetro llamado nombre. Al llamar a la función saludar("Juan"), se está sustituyendo el argumento "Juan" en lugar del parámetro nombre dentro de la función. Como resultado, la función imprimirá "¡Hola, Juan!" en la consola.

Es importante tener en cuenta que los argumentos pueden ser de diferentes tipos, como cadenas de texto, números, listas, diccionarios, entre otros. Además, es posible pasar múltiples argumentos separados por comas al llamar a una función, siempre y cuando se correspondan con los parámetros definidos en la función.

Por ejemplo:

```
def sumar(a, b):  
    resultado = a + b  
    print("El resultado de la suma es:", resultado)  
  
# Llamada a la función con dos argumentos  
sumar(3, 5)
```

En este caso, la función sumar recibe dos argumentos: a y b. Al llamar a la función sumar(3, 5), se sustituyen los argumentos 3 y 5 en lugar



de los parámetros a y b, respectivamente. La función imprimirá "El resultado de la suma es: 8" en la consola.

La sustitución de argumentos o parámetros es fundamental para que las funciones puedan recibir datos específicos y realizar operaciones basadas en ellos. Permite que las funciones sean más flexibles y reutilizables, ya que se pueden llamar con diferentes valores en diferentes contextos.

## 2.5 Ámbito: variables locales y globales

El ámbito en Python se refiere al alcance o contexto en el que una variable es válida y puede ser accedida. En Python, existen dos tipos principales de ámbito: variables locales y variables globales.

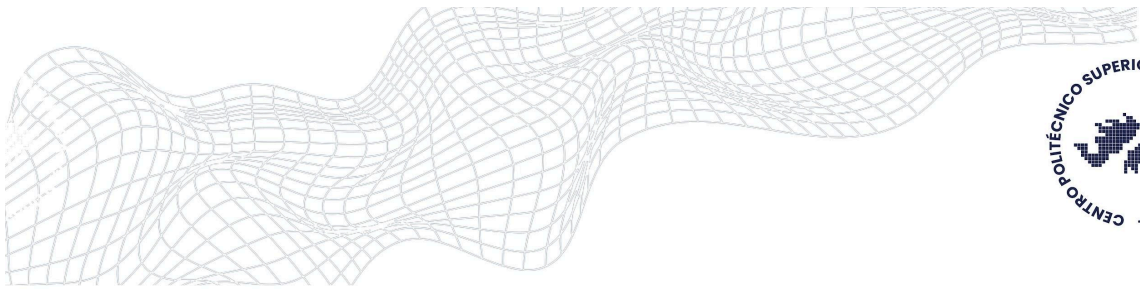
### 2.5.1 Variables locales

Las variables locales son aquellas que se definen dentro de una función y solo pueden ser accedidas desde dentro de esa función. Estas variables son visibles y utilizables sólo dentro del bloque de código en el que se definen. Una vez que la función termina su ejecución, las variables locales se eliminan de la memoria.

#### Aquí tienes un ejemplo de una variable local

```
def mi_funcion():  
    x = 10 # Variable local  
    print(x)
```





```
mi_funcion() # Salida: 10  
print(x) # Error: NameError: name 'x' is not defined
```

En este caso, la variable `x` es una variable local que se define dentro de la función `mi_funcion()`. Solo se puede acceder a ella dentro de la función y no fuera de ella. Al tratar de acceder a `x` fuera de la función, se generará un error.

### 2.5.2 Variables globales

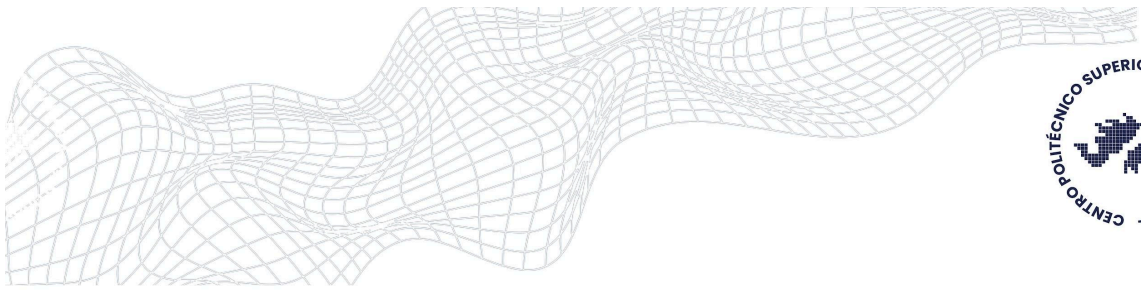
Las variables globales son aquellas que se definen fuera de cualquier función o bloque de código y están disponibles en todo el programa. Estas variables pueden ser accedidas y utilizadas desde cualquier parte del código, ya sea dentro o fuera de las funciones.

Aquí tienes un ejemplo de una variable global:

```
x = 10 # Variable global  
  
def mi_funcion():  
    print(x)  
  
mi_funcion() # Salida: 10  
print(x) # Salida: 10
```

En este caso, la variable `x` es una variable global que se define fuera de cualquier función. Puede ser accedida tanto dentro de la función





`mi_funcion()` como fuera de ella, ya que su ámbito se extiende a todo el programa.

Es importante tener en cuenta que, si se define una variable local con el mismo nombre que una variable global, la variable local tendrá precedencia dentro del ámbito de la función. Esto se conoce como "enmascaramiento" de la variable global.

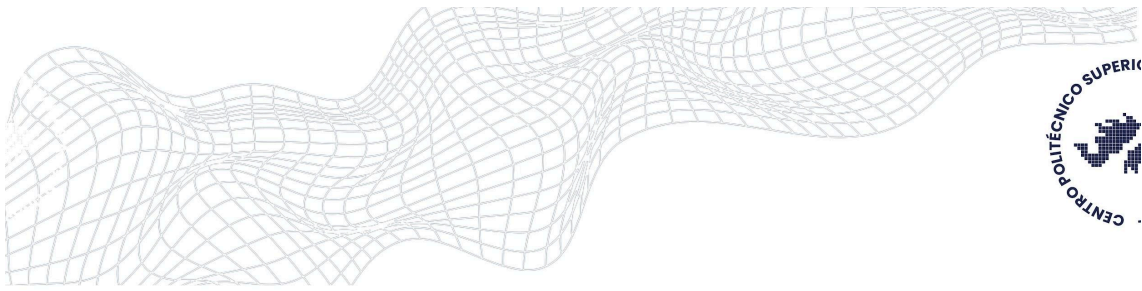
```
x = 10 # Variable global

def mi_funcion():
    x = 5 # Variable local que enmascara a la variable global
    print(x)

mi_funcion() # Salida: 5
print(x) # Salida: 10
```

En este ejemplo, la variable local `x` dentro de la función `mi_funcion()` enmascara a la variable global `x`. Por lo tanto, dentro de la función, se imprimirá el valor de la variable local (5), mientras que fuera de la función, se imprimirá el valor de la variable global (10).

Es importante entender y tener en cuenta el ámbito de las variables en Python para evitar errores y asegurarse de acceder a las variables adecuadas en el contexto correcto.



## Ejercicio análisis práctico



Observa esta función que modifica una variable global. Pensá: ¿qué efecto tiene usar una variable global dentro de una función? ¿Qué puede salir bien... y qué no tanto?

```
x = 5 # Variable global

def modificar_global():
    global x
    x = 10 # Modificando la variable global dentro de la función

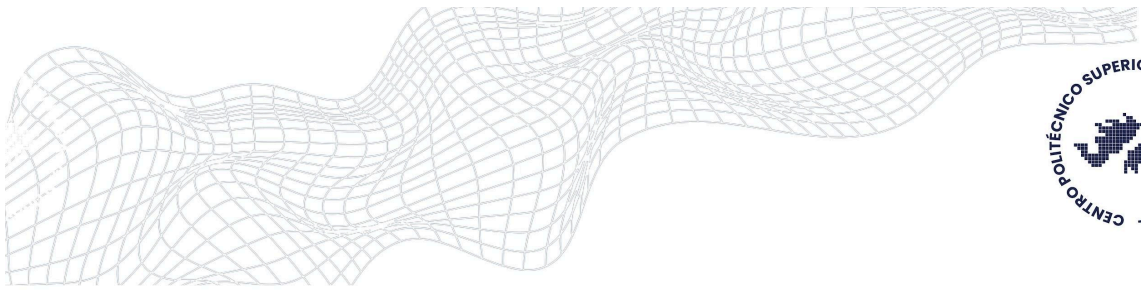
modificar_global()
print(x) # ¿Qué imprimirá?
```

## Pregunta para seguir reflexionando...

- ¿Es realmente una buena práctica modificar variables globales desde dentro de una función? Argumenta tu respuesta: ¿por qué sí o por qué no?

Y por último:

- ¿Qué otras estrategias podrías usar para evitar depender de variables globales, especialmente en programas más complejos?



## 2.6 Recursión (recursividad)

La recursión, también conocida como recursividad, es un concepto en programación que se refiere a la capacidad de una función de llamarse a sí misma directa o indirectamente. En Python, se puede utilizar la recursión para resolver problemas dividiéndolos en subproblemas más pequeños y resolviendo cada subproblema de manera recursiva.

La recursión se basa en dos conceptos fundamentales:

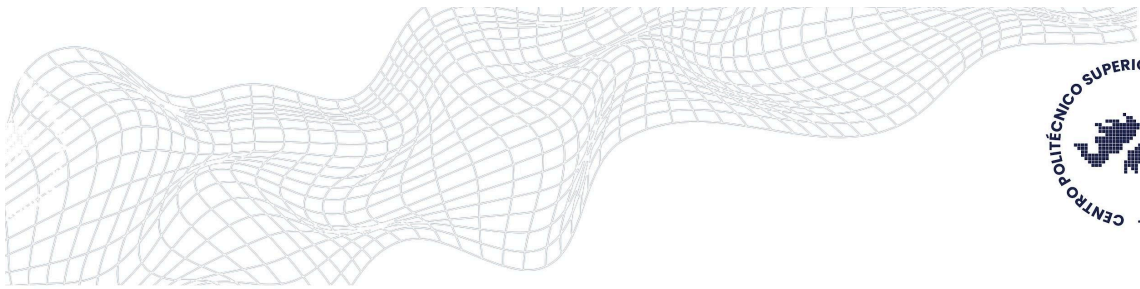
1. Caso base:

Es el punto de terminación de la recursión. Define una condición que se evalúa dentro de la función recursiva y determina cuándo se debe detener la llamada recursiva. El caso base evita que la función se llame infinitamente y asegura que la recursión se detenga en algún momento.

2. Caso recursivo:

Es la parte donde la función se llama a sí misma para resolver un problema más pequeño o similar al original. En cada llamada recursiva, el problema se simplifica o se reduce en tamaño hasta que se alcance el caso base.

Aquí tienes un ejemplo sencillo para ilustrar la recursión en Python:

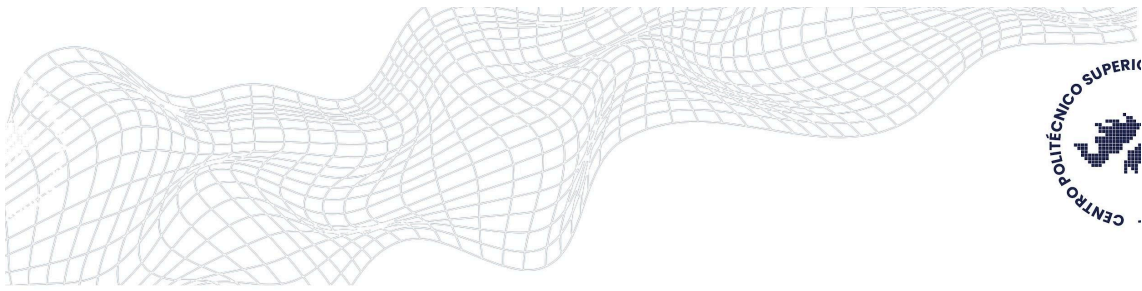


```
def contar_hasta_cero(n):  
    if n <= 0: # Caso base  
        print("¡Boom!")  
    else: # Caso recursivo  
        print(n)  
        contar_hasta_cero(n - 1) # Llamada recursiva  
  
contar_hasta_cero(5)
```

En este caso, la función `contar_hasta_cero()` cuenta desde un número dado `n` hasta cero. El caso base se verifica cuando es menor o igual a cero, en cuyo caso se imprime "¡Boom!". En el caso recursivo, se imprime el valor actual de `n` y luego se llama a la función `contar_hasta_cero()` con `n - 1` para continuar el proceso de contar hacia cero.

```
5  
4  
3  
2  
1  
¡Boom!
```

Es importante tener en cuenta que la recursión debe utilizarse con precaución, ya que puede consumir mucha memoria y tiempo de ejecución si no se implementa correctamente. Un error común en la



recursión es olvidar definir un caso base o no asegurarse de que el caso recursivo converja hacia el caso base. Esto puede provocar un bucle infinito y agotar los recursos del sistema.



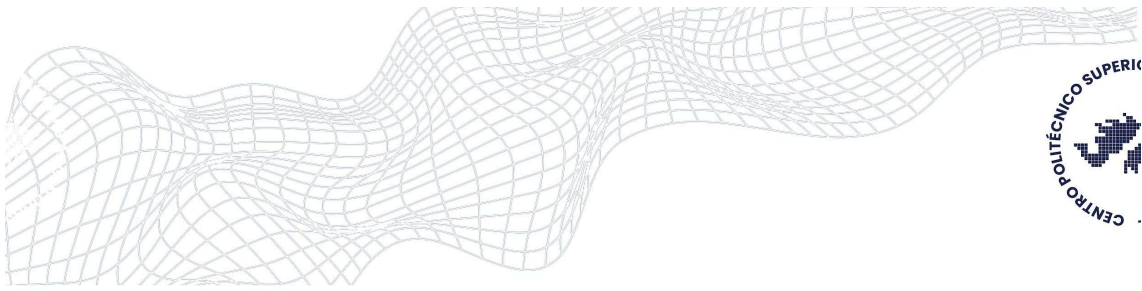
### **Para pensar un poco...**

Cuando usás recursión,

¿Qué es lo que evita que la función quede atrapada en un bucle infinito?

¿Qué condición tiene que cumplirse sí o sí para que la función termine como corresponde?

Además, ¿qué ventaja te da la recursión frente a una solución iterativa? Pensá en esos casos donde una solución recursiva puede ser más clara, más elegante o directamente más práctica.



## Actividad de cierre



Vamos a realizar una actividad para entender mejor cómo funcionan las funciones en Python y cómo se pueden usar para organizar el código de manera clara y reutilizable.

### PASO 1 >> Usamos la IA para generar un ejemplo

Abrí Chat GPT y escribí:

💬 ***"Genera una función en Python que calcule el promedio de una lista de números y que incluya un docstring explicativo."***

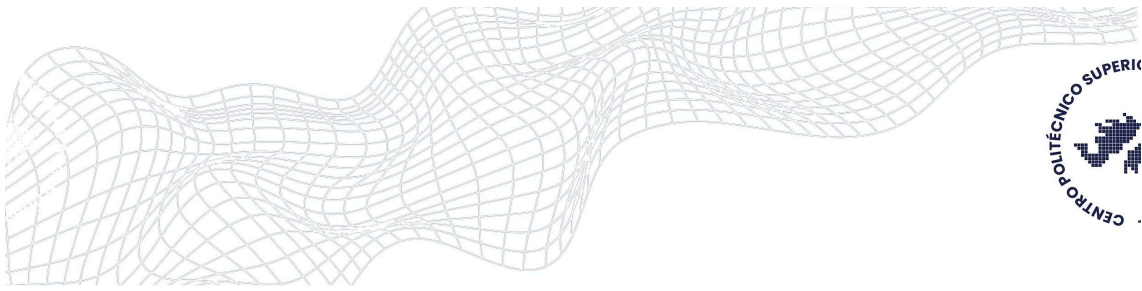
Copió el código que te da y léelo con atención.

### PASO 2 >> Analizamos el código

Explicá con tus palabras **qué hace el código, línea por línea**.

Luego, pensá: ¿cómo podrías resolver lo mismo sin usar una función?, ¿te parece más fácil o más complicado?




👉 **Importante:** Esta parte hacela vos, sin usar Chat GPT. Queremos ver cómo lo entendiste y cómo aplicas los conceptos vistos hoy (definición, parámetros, return, etc.).



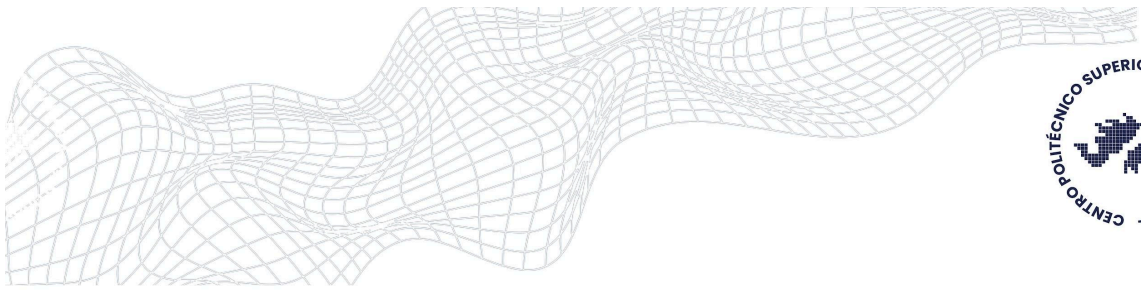
### PASO 3 >> Compartimos en el foro



Subí al foro de intercambio de la Clase N°6 lo siguiente:

1.  El código que te generó ChatGPT.
2.  Tu explicación línea por línea.
3.  Una alternativa sin usar funciones (si se te ocurre).





## Ejercicios de práctica con funciones y procedimientos



A continuación, resolvé estos tres ejercicios usando funciones en Python y todo lo aprendido en esta clase:

### 1. Suma de elementos de una lista

Creá una función llamada `sumar_lista` que reciba una lista de números y devuelva la suma de todos sus elementos.

-> Probá usarla con esta lista: `[3, 7, 1, 10]`.

### 2. Calcular el factorial de un número (recursivo)

Creá una función llamada `factorial` que reciba un número entero y devuelva su factorial.

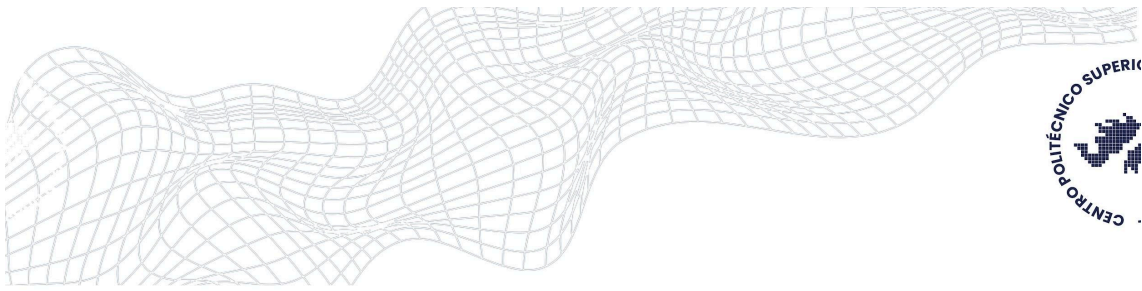
Recordá que el factorial de un número es el resultado de multiplicar ese número por todos los anteriores hasta llegar a 1.

-> Por ejemplo: el factorial de 4 es  $4 * 3 * 2 * 1 = 24$ .

### 3. Ordenar letras de una palabra

Hacé una función que se llame `ordenar_cadena`, que reciba una palabra como texto y devuelva esa misma palabra con las letras ordenadas alfabéticamente (respetando mayúsculas y minúsculas).

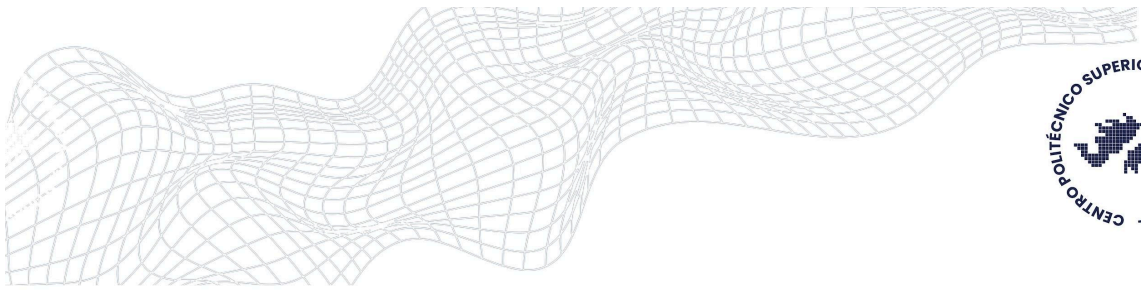
-> Probá con esta palabra: "PythonEsDivertido".



### 3. Cierre

En esta clase exploramos los pilares de la programación estructurada mediante el uso de funciones y procedimientos. Aprendimos cómo dividir un programa en partes más pequeñas, reutilizables y organizadas, lo cual mejora la legibilidad, facilita el mantenimiento y reduce errores. También nos adentramos en la recursividad, una técnica poderosa para resolver problemas complejos descomponiéndose en subproblemas similares.

Estos conceptos no solo nos permiten escribir código más claro y modular, sino que representan una forma de pensar la programación como un conjunto de bloques que cooperan entre sí. Comprender el paso de argumentos, el ámbito de las variables y cómo se invocan funciones es clave para avanzar hacia programas más robustos, escalables y eficientes.



## 4. Bibliografía



- Matthes, E. (2019). Python crash course: A hands-on, project-based introduction to programming (2.ª ed.). No Starch Press.
- Lutz, M. (2013). Learning Python (5.ª ed.). O'Reilly Media.
- Beazley, D., & Jones, B. K. (2013). Python cookbook: Recipes for mastering Python 3 (3.ª ed.). O'Reilly Media.
- Ramalho, L. (2015). Fluent Python: Clear, concise, and effective programming. O'Reilly Media.
- OpenAI. (2025). ChatGPT (versión GPT-4). Recuperado de <https://chat.openai.com>