**UNIVERSIDAD PONTIFICIA BOLIVARIA SECCIONAL BUCARAMANGA**

**FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMÁTICA**

**LÓGICA DE PROGRAMACIÓN**

**JUAN JOSÉ DÍAZ PRADA**

# **Funciones**

(Lab 2.4)

Objetivos:

* El estudiante estará en capacidad aplicar los conceptos estudiados en clase sobre el uso de las funciones para resolver problemas con el lenguaje Python.
* El estudiante participará activamente en clase en la solución de los problemas planteados, compartiendo código, explicaciones y opiniones sobre las soluciones propuestas en la clase.
* El estudiante trabajará organizadamente todo su código fuente en su repositorio de GitHub y comentará este código fuente.

**Funciones en Python**

Las funciones son bloques de código que se pueden reutilizar simplemente llamando a la función. Esto permite la reutilización de código simple y elegante sin volver a escribir explícitamente secciones de código. Esto hace que el código sea más legible, facilita la depuración y limita los errores de escritura.

Las funciones en Python son componentes importantes en la programación que cuentan con una estructura que consta de dos principios:

👉🏻 **Principio de reutilización:** Puedes reutilizar una función varias veces y en distintos programas.

👉🏻 **Principio de modularidad:** Te permite segmentar programas complejos con módulos más simples para depurar, programar con mayor facilidad, así como aumentar la legibilidad del código resultante.

### **Sentencia def**

Esta definición de función se usa para crear objetos, las cuales son definidas por cada programador. Es una sentencia para ejecutar con el nombre de la función y tienen referencias al nombre de esta.

La sintaxis de la función en Python se describe a continuación y se explica el significado de cada elemento.

👉🏻 *NOMBRE*: es asignado por cada programador, recuerda ser acertado y descriptivo.

👉🏻 *LISTA\_DE\_PARAMETROS*: la lista que incluye la función.

👉🏻 *DOCSTRING\_DE\_FUNCION*: cadena de caracteres para la función.

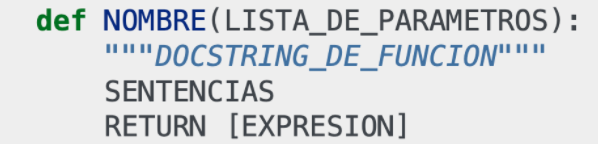
👉🏻 *SENTENCIAS*: es el bloque de códigos que se realiza en cada operación Python.

👉🏻 *RETURN*: es la sentencia return en el código.

👉🏻 *EXPRESION*: la variable que regresa la sentencia RETURN.

La instrucción Pass es como lo indica su nombre una expresión nula, no hace nada. Es casi como si no existiera, pero nos permite crear un bucle sin colocar código en su cuerpo para añadirlo más tarde utilizándolo como un relleno temporal.

La diferencia con *continue* es que este termina la iteración actual pero continua con la siguiente instrucción o iteración del bucle, volviendo al inicio. En cambio, *pass* no hace nada y continua con las siguientes instrucciones de este sin volver al inicio.

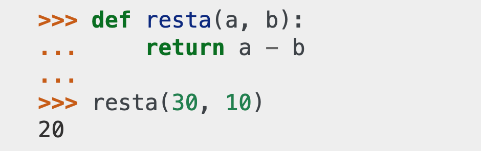
****

### **Argumentos y parámetros**

La definición de una función en Python con valores, denominados parámetros, se pueden convertir en argumentos cuando interviene una llamada de los valores.

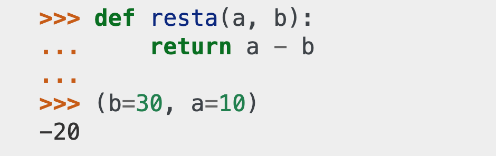
**Por posición**

En las funciones de Python, cuando se envían argumentos en las funciones se reciben por orden.



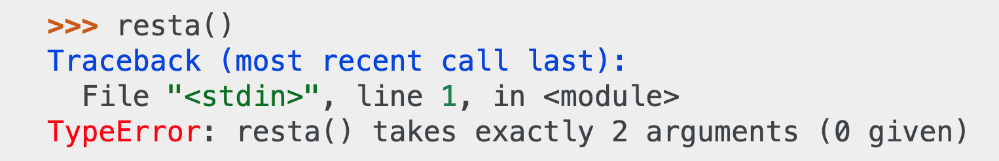
**Por nombre**

También es posible evitar el orden de los parámetros si es que durante la llamada se especifica el valor de cada uno según su nombre.



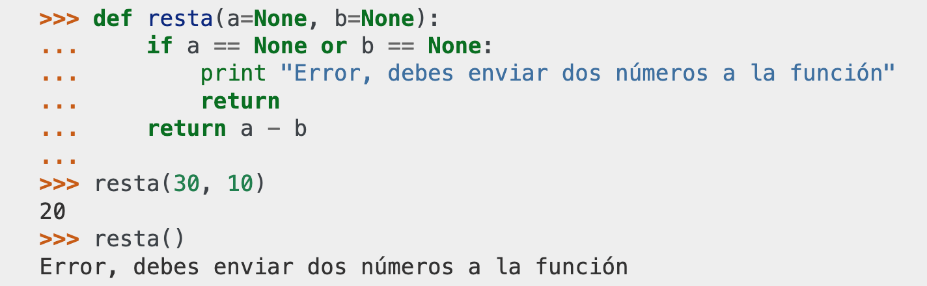
**Llamada sin argumentos**

Cuando se llama una función con los parámetros respectivos, esta puede no pasar los argumentos de manera correcta y esto provocará una excepción *TypeError.*



**Parámetros por defecto**

Si quieres solucionar este error, debes asignar valores por defecto nulos (None) a los respectivos parámetros. De esta manera puedes prever y comprobar el funcionamiento sin antes ejecutar el código.

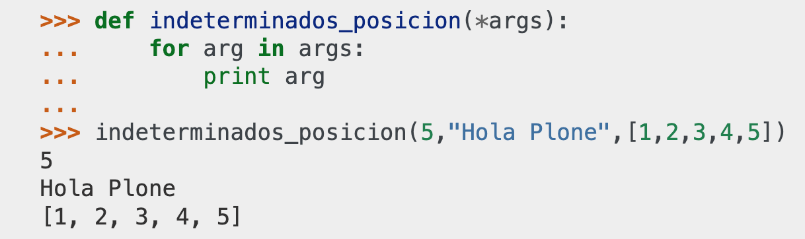


### **Argumentos indeterminados**

Dentro de las funciones, Python brinda la opción de utilizar parámetros indeterminados por posición y nombre. Esto te ayuda en los momentos en donde no has definido la cantidad de elementos que necesitas para tu función.

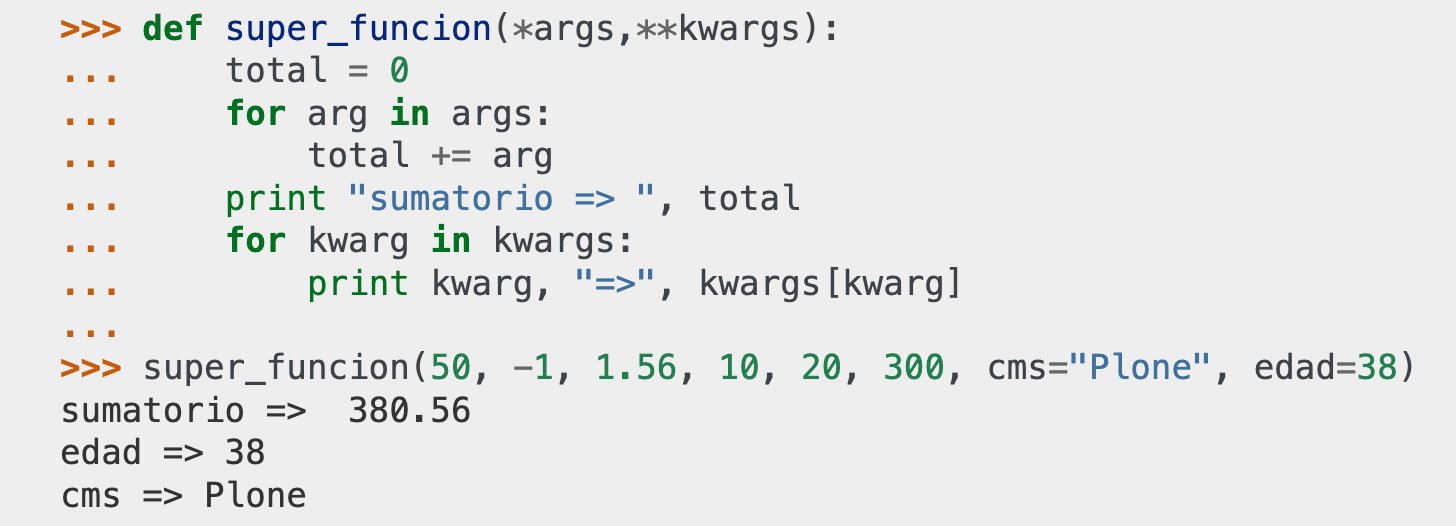
**Por posición**

En esta función debes crear una lista dinámica de argumentos. Esto se representa como un tipo dupla que se define con un asterisco.



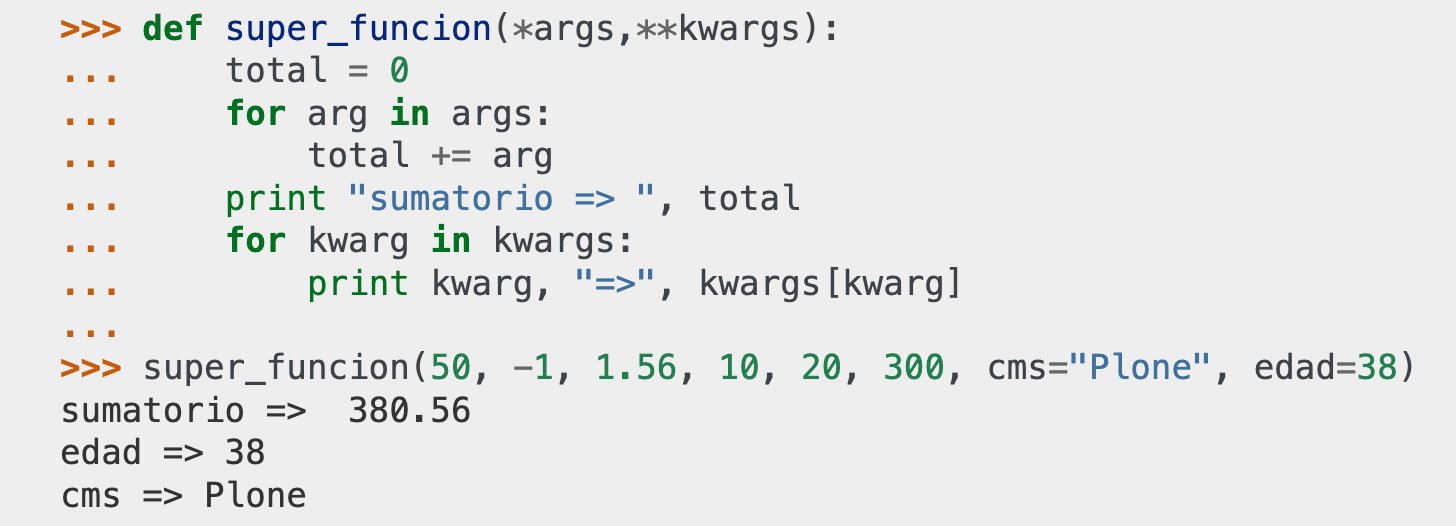
**Por nombre**

En esta función debes crear un diccionario dinámico de argumentos para definir el parámetro con dos asteriscos.



**Por posición y nombre**

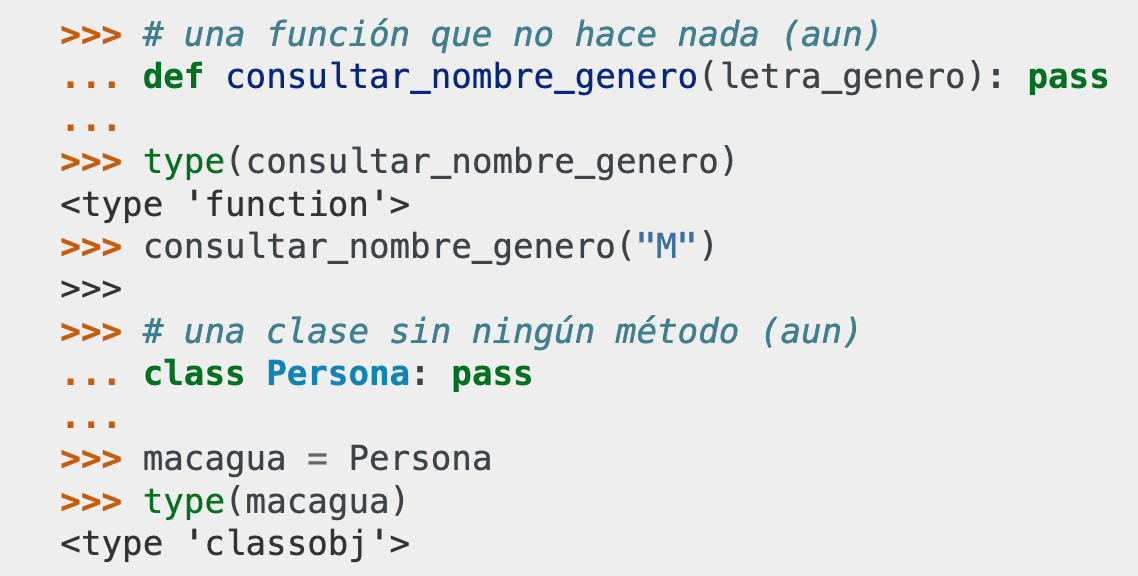
En esta función debes crear colecciones dinámicas para ambos tipos de parámetros de manera simultánea. Incluyes los argumentos por valor y luego por clave. No es necesario que coloques args y kwargs, solo en caso de convención.



### **Sentencia pass**

Las funciones en Python te permiten efectuar una operación nula, es decir que cuando se emplea no sucede nada. Esto te puede ayudar cuando una sentencia se requiere de manera sintáctica y no necesita de un código para efectuarse.

Por ahora, esa sentencia en combinación con funciones se considera de nivel avanzado. A continuación, un ejemplo:

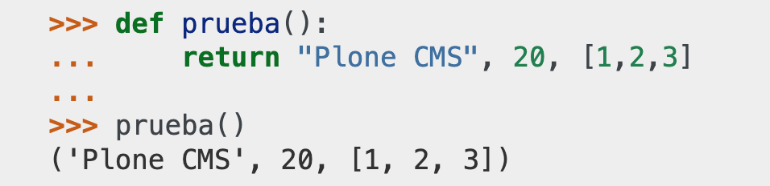


### **Sentencia return**

La sentencia return te ayuda a que las funciones se puedan comunicar con el exterior. Esto se da gracias a la devolución de valores.

**Retorno múltiple**

En Python tienes la posibilidad de retornar valores múltiples separados en comas. A su vez, se puede registrar a distintas variables y valores de la tupla inmutable.



Aquí encuentras la documentación oficial de Python sobre funciones:

[Funciones Built-in — documentación de Python - 3.9.4](https://docs.python.org/es/3/library/functions.html#len)

**¿Cómo crear funciones en Python?**

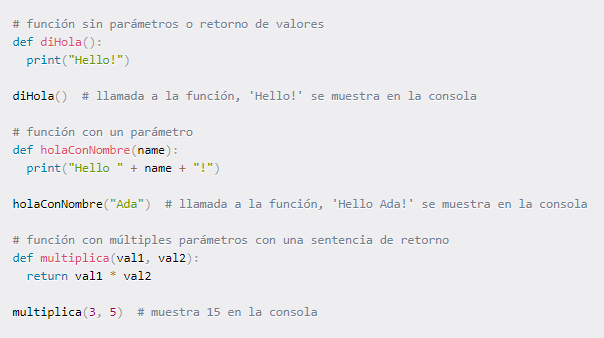
Después de comprender todas las funciones en Python vamos a mirar cómo se construyen en Python. 🧐

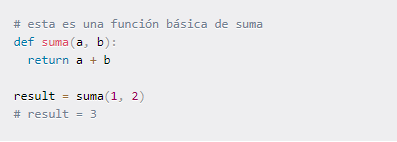
### **Sintaxis**

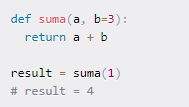
En Python, una definición de función tiene las siguientes características:

1. La palabra clave def
2. Un nombre de función
3. Paréntesis ’()’, y dentro de los paréntesis los parámetros de entrada, aunque los parámetros de entrada sean opcionales.
4. Dos puntos ’:’
5. Algún bloque de código para ejecutar
6. Una sentencia de retorno (opcional)

Ejemplos:

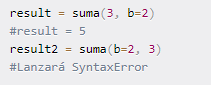




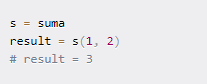




1. No es posible pasar un argumento de palabra clave antes que uno que no sea de palabra clave:



1. Las funciones también son objetos, por lo que puedes asignarlas a una variable y usar esa variable como una función.



**Escribe los siguientes códigos en tu PC y discute las salidas con tus compañeros y profesora, no olvides ir anexando captura de las salidas a tu informe:**

1. **Función max()**

max() es una función incorporada en Python 3. Devuelve el elemento más grande en un iterable o el más grande de dos o más argumentos.

**Argumentos**

Esta función toma dos o más números o cualquier tipo de iterable como argumento. Mientras damos un iterable como argumento, debemos asegurarnos de que todos los elementos del iterable sean del mismo tipo. Esto significa que no podemos pasar una lista que tenga almacenados valores enteros y de cadena.

**Sintaxis:** max (iterable, \* iterables [, clave, predeterminado]) max (arg1, arg2, \* args [, clave])

**Argumentos válidos:**

max(2, 3)

max([1, 2, 3])

max('a', 'b', 'c')

**Argumentos inválidos:**

max(2, 'a')

max([1, 2, 3, 'a'])

max([])

****

**Valor devuelto**

Se devuelve el elemento más grande del iterable. Si se proporcionan dos o más argumentos posicionales, se devuelve el mayor de los argumentos posicionales. Si el iterable está vacío y no se proporciona el valor predeterminado, se genera un *ValueError*.



**Escribe al frente de cada expresión la evaluación de la salida obtenida, hazlo como un comentario.**

print(max(2, 3)) # 3, 3 es mayor a 2

print(max(2, 3, 23)) # 23, es mayor a 2 y 3

list1 = [1, 2, 4, 5, 54]

print(max(list1)) # Devuelve 54 ya que 54 es el valor más grande de la lista

list2 = ['a', 'b', 'c’]

print(max(list2)) # Devuelve c, ya que es el mayor en el abecedario

list3 = [1, 2, 'abc', 'xyz']

print(max(list3)) # Se genera TypeError ya que los valores en la lista son de tipos diferentes

#Soluciona el TypeError mencionado anteriormente antes de continuar con la siguiente evaluación de funciones.

Se remueven los valores 1 y 2, la salida es xyz

list4 = []

print(max(list4)) # Se genera ValueError, la lista está vacia.

1. **Función min ()**

**min()** es una función incorporada en Python 3. Devuelve el elemento más pequeño en un iterable o el más pequeño de dos o más argumentos.

**Argumentos**

Esta función toma dos o más números o cualquier tipo de iterable como argumento. Mientras damos un iterable como argumento, debemos asegurarnos de que todos los elementos del iterable sean del mismo tipo. Esto significa que no podemos pasar una lista que tenga almacenados valores enteros y de cadena.

**Argumentos válidos:**

min(2, 3)

min([1, 2, 3])

min('a', 'b', 'c')

**Argumentos inválidos:**

min(2, 'a')

min([1, 2, 3, 'a'])

min([])



**Valor devuelto**

Se devuelve el elemento más pequeño del iterable. Si se proporcionan dos o más argumentos posicionales, se devuelve el más pequeño de los argumentos posicionales. Si el iterable está vacío y no se proporciona el valor predeterminado, se genera un *ValueError.*



**Escribe al frente de cada expresión la evaluación de la salida obtenida, hazlo como un comentario.**

print(min(2, 3)) # Devuelve 2 ya que 2 es el más pequeño de los dos valores

print(min(2, 3, -1)) # Devuelve -1, es menor que 2 y 3

list1 = [1, 2, 4, 5, -54]

print(min(list1)) # Devuelve -54 ya que -54 es el más pequeño de los valores en la lista.

list2 = ['a', 'b', 'c']

print(min(list2)) # Devuelve a, ya que va primero que b y c en el abecedario

list3 = [1, 2, 'abc', 'xyz']

print(min(list3)) # Se genera TypeError, ya que los valores no son compatibles

#Soluciona el TypeError mencionado anterioremente antes de continuar con el siguiente paso

Se elimina 1 y 2, el resultado es “abc”

list4 = []

print(min(list4)) # Se genera ValueError ya que el argumento está vacío

1. **Función divmod()**

**divmod()** es una función incorporada en Python 3, que devuelve el cociente y el resto al dividir el número a por el número b. Toma dos números como argumentos a & b. El argumento no puede ser un número complejo.

**Argumentos**

Se necesitan dos argumentos a & b: un número entero o un número decimal. No puede ser un número complejo.



**Valor devuelto**

El valor de retorno será el par de números positivos que consiste en el cociente y el resto obtenido al dividir a por b. En el caso de tipos de operandos mixtos, se aplicarán las reglas para los operadores aritméticos binarios.

Para los argumentos de números enteros, el valor de retorno será el mismo que (a // b, a % b).

Para los argumentos de números decimales, el valor devuelto será el mismo que (q, a % b), donde q es normalmente math.floor (a / b) (La función**math.floor** devuelve el número entero más próximo al pasado como argumento que sea menor o igual a él) pero puede ser 1 menos que eso.



**Escribe al frente de cada expresión la evaluación de la salida obtenida, hazlo como un comentario.**

print(divmod(5,2)) # muestra (2,1)

print(divmod(13.5,2.5)) # Muestra (5.0, 2.0)

q, r = divmod(13.5,2.5) # Asigna q=cociente & r=resto

print(q) # muestra 5.0 porque math.floor(13.5/2.5) = 5.0

print(r) # Muestra 2.0, ya que esto es lo que sobra de la division

**Consulta tres funciones de Python y explícalas según el formato anterior, breve descripción, argumentos, valor devuelto y códigos de ejemplo.**

**Anexa a tu informe de laboratorio 2.4 esa consulta y comparte en el espacio para la clase designado para ello con tus compañeros esa consulta.**

No olvides actualizar tu Git con todas los código que realizas en clase, ya sea los básicos compartidos por la profesora o las mejoras que tú o tus compañeros proponen, eso te sirve de práctica y para la apropiación de la sintaxis del lenguaje.

**4.1. Función round()**

La función round se encarga de redondear el número digitado a n dígitos.

**Argumentos**

round() se compone de dos argumentos, el número a redondear y el aproximado que se quiere. Pueden ser tanto integer como float.

round(6, 10)

round(0.8, 1)

round(2.6)

**Valor devuelto**

round() devuelve el número mas cercano a la potencia menos n dígitos. Si n dígitos se omite, se redondea al valor más cercano. Si dos valores están igual de cerca, se redondea al valor par.





**4.2. Función bin()**

Transforma números enteros en una cadena binaria

**Argumentos**

Cualquier integer, puede ser transformado en binario. Si se quiere utilizar un float, se debe especificar al programa que lo transforme en integer primero

bin(15)

bin(32)

bin(int(3.5)

**Valor devuelto**

El programa transforma el numero digitado en cadena binaria y lo devuelve con el prefijo “0b”





**4.3. Función sum()**

Suma los elementos de un iterable de izquierda a derecha

**Argumentos**

Utilizado en listas numéricas, sum() no puede sumar cadenas

Hola = [4, 6]

sum(hola)

**Valor devuelto**

El programa procede a sumar todos los valores dentro de la lista y arroja el resultado.



