**Funciones en Python**

Actividad para su desarrollo paso a paso desde consola

### **¿Qué es una función?**

Una función, no es más que un bloque de código el cual posee un nombre y recibe algunos argumentos de entrada o ninguno, para poder realizar una determinada operación y al final nos retorna un resultado.

Lo mejor de todo, es que este bloque lo podemos llamar todas las veces que queramos.

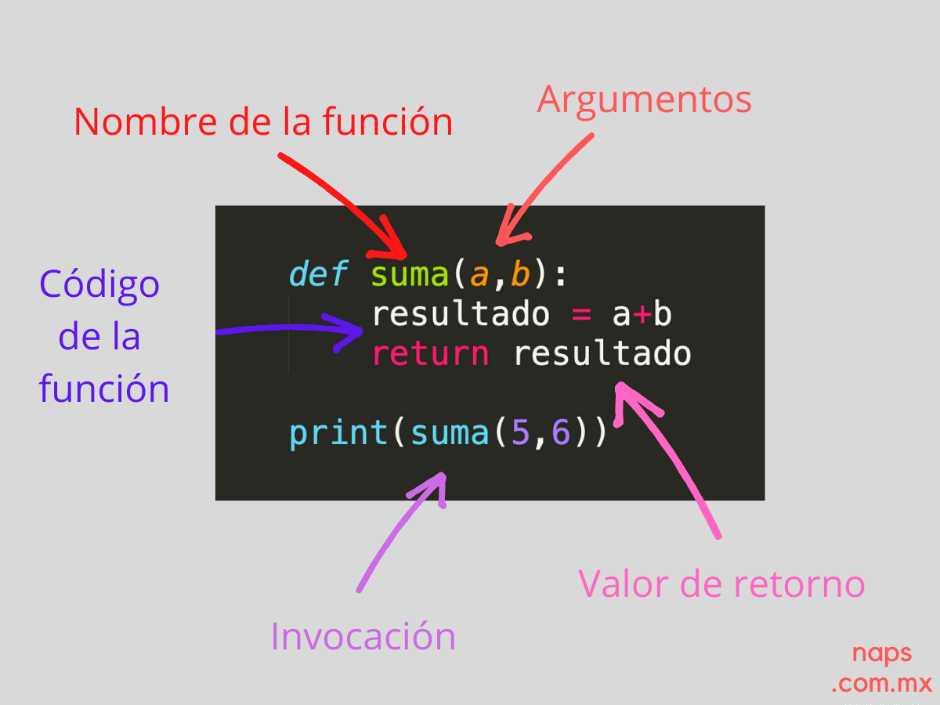
## **Introducción a las funciones en Python**

Una función te permite definir un bloque de código reutilizable que se puede ejecutar muchas veces dentro de tu programa.

Las funciones te permiten crear soluciones más modulares y [DRY](https://en.wikipedia.org/wiki/Don%27t_repeat_yourself) para problemas complejos.

Si bien Python ya proporciona muchas funciones integradas como print() y len(), también puedes definir tus propias funciones para usar en tus proyectos.

Una de las grandes ventajas de usar funciones en tu código es que reduce el número total de líneas de código en tu proyecto.



### **Sintaxis**

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

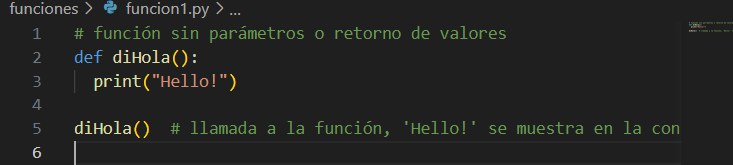
1. La palabra clave def
2. Un nombre de función
3. Paréntesis ’()’, y dentro de los paréntesis los parámetros de entrada, aunque los parámetros de entrada sean opcionales.
4. Dos puntos ’:’
5. Algún bloque de código para ejecutar
6. Una sentencia de retorno (opcional)

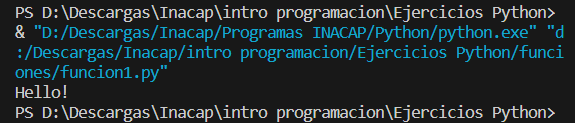
# función sin parámetros o retorno de valores

def diHola():

print("Hello!")

diHola() # llamada a la función, 'Hello!' se muestra en la consola



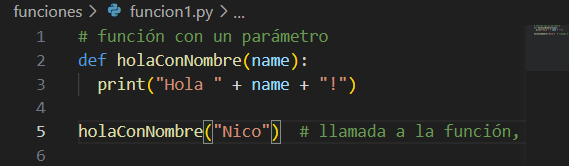


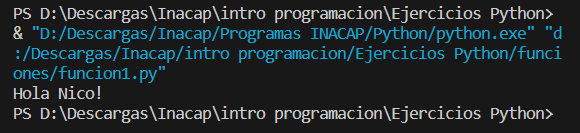
# función con un parámetro

def holaConNombre(name):

print("Hola " + name + "!")

holaConNombre("Pablo") # llamada a la función, “Hola Pablo!” se muestra en la consola



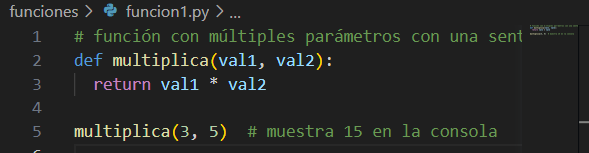


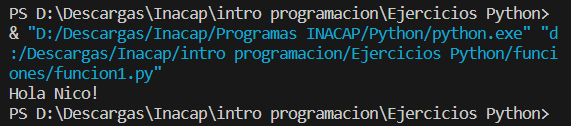
# función con múltiples parámetros con una sentencia de retorno

def multiplica(val1, val2):

return val1 \* val2

multiplica(3, 5) # muestra 15 en la consola





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 se crean usando la palabra clave def, seguida de un nombre de función y parámetros de función entre paréntesis.

Una función siempre devuelve un valor. La función utiliza la palabra clave return  para devolver un valor; si no desea devolver ningún valor, se devolverá el valor predeterminado None.

El nombre de la función se usa para llamar a la función, pasando los parámetros necesarios entre paréntesis.

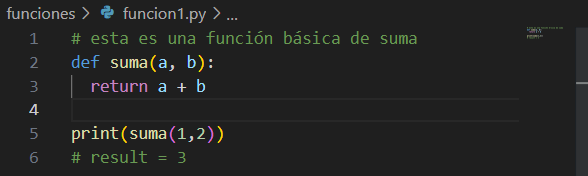
# esta es una función básica de suma

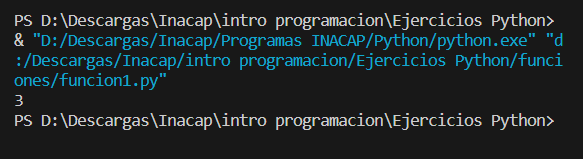
def suma(a, b):

return a + b

result = suma(1, 2)

# result = 3





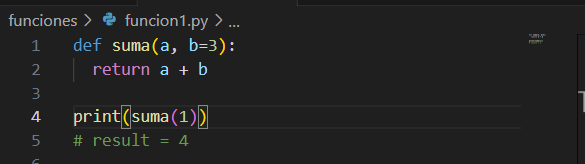
Puedes definir valores predeterminados para los parámetros, de esa manera Python interpretará que el valor de ese parámetro es el predeterminado si no se proporciona ninguno.

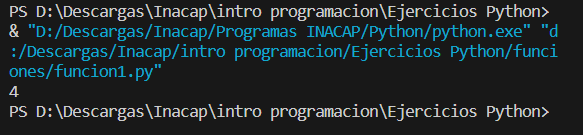
def suma(a, b=3):

return a + b

result = suma(1)

# result = 4





Puedes pasar los parámetros en el orden que desees, utilizando el nombre del parámetro.

result = suma(b=2, a=2)

# result = 4

Sin embargo, no es posible pasar un argumento de palabra clave antes que uno que no sea de palabra clave

result = suma(3, b=2)

#result = 5

result2 = suma(b=2, 3)

#Lanzará SyntaxError

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

s = suma

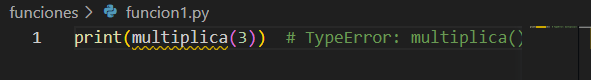
result = s(1, 2)

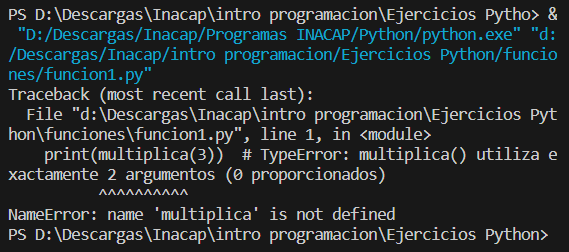
# result = 3

### **Notas**

Si la definición de una función incluye parámetros, debe proporcionar el mismo número de parámetros cuando llame a la función.

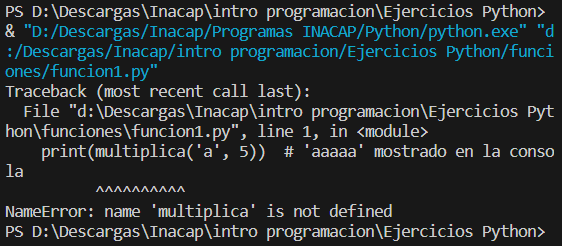
print(multiplica(3)) # TypeError: multiplica() utiliza exactamente 2 argumentos (0 proporcionados)





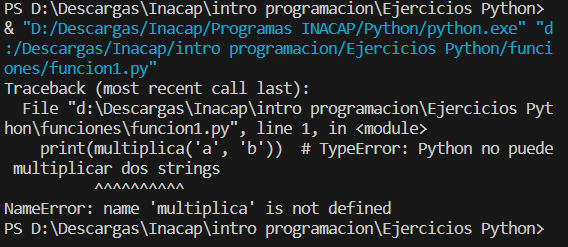
print(multiplica('a', 5)) # 'aaaaa' mostrado en la consola





print(multiplica('a', 'b')) # TypeError: Python no puede multiplicar dos strings





El bloque de código que ejecutará la función incluye todas las declaraciones con indentación dentro de la función.

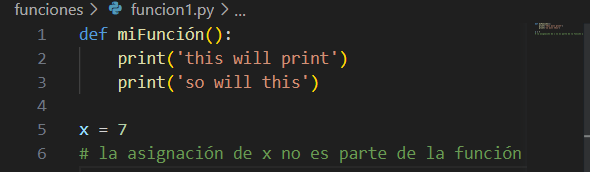
def miFunción():

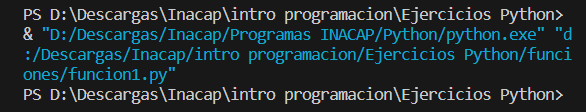
print('this will print')

print('so will this')

x = 7

# la asignación de x no es parte de la función ya que no está indentada





Las variables definidas dentro de una función solo existen dentro del ámbito de esa función.

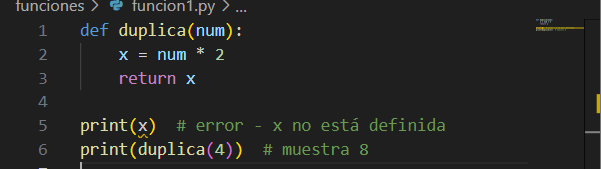
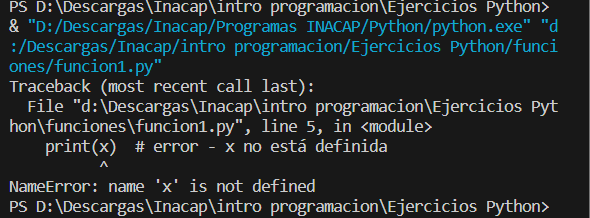
def duplica(num):

x = num \* 2

return x

print(x) # error - x no está definida

print(duplica(4)) # muestra 8

Python interpreta el bloque de funciones solo cuando se llama a la función y no cuando se define la función. Por lo tanto, incluso si el bloque de definición de función contiene algún tipo de error, el intérprete de Python lo señalará solo cuando se llame a la función.

Ahora veamos algunas funciones específicas con ejemplos.

## 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([])

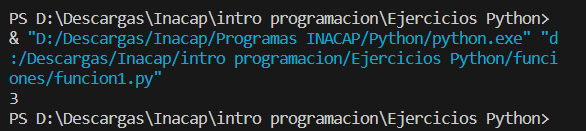
### Valor devuelto

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.

### Código de ejemplo:

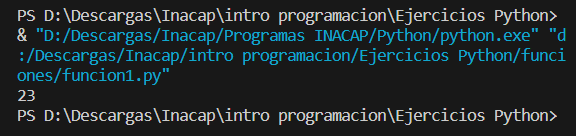
print(max(2, 3)) # Devuelve 3 ya que 3 es el mayor de los dos valores





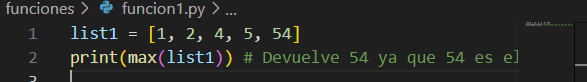
print(max(2, 3, 23)) # Devuelve 23 ya que 23 es el mayor de todos los valores

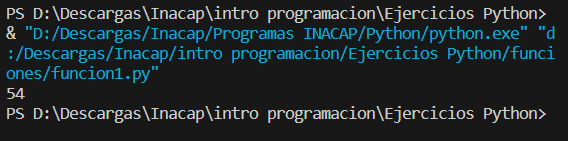




list1 = [1, 2, 4, 5, 54]

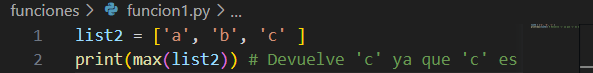
print(max(list1)) # Devuelve 54 ya que 54 es el valor más grande de la lista

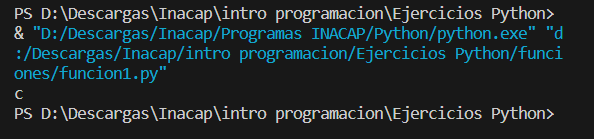




list2 = ['a', 'b', 'c' ]

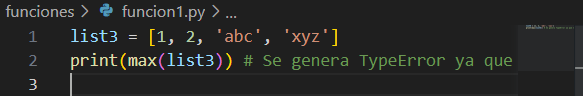
print(max(list2)) # Devuelve 'c' ya que 'c' es el valor más grande en la lista porque 'c' tiene un valor ascii mayor que 'a' ,'b'.

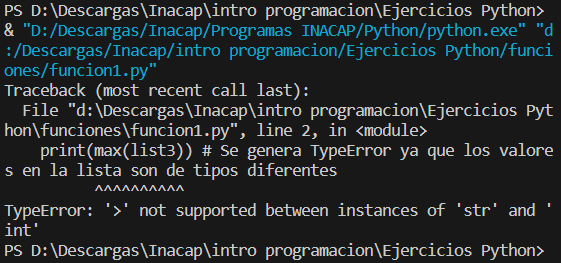




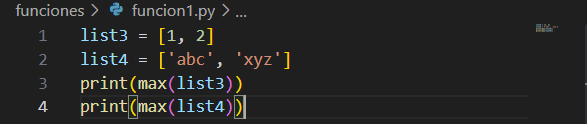
list3 = [1, 2, 'abc', 'xyz']

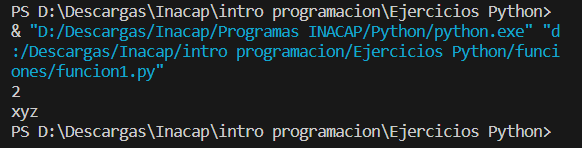
print(max(list3)) # Se genera TypeError ya que los valores en la lista son de tipos diferentes





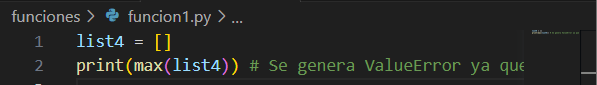
#Soluciona el TypeError mencionado anterioremente antes de continuar con el siguiente paso

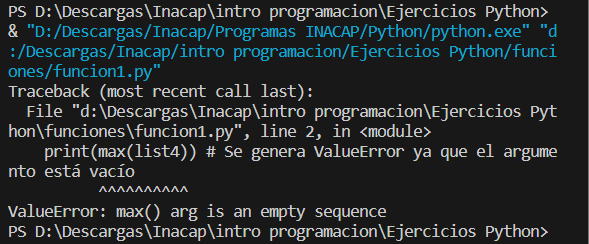




list4 = []

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





[Documentación oficial](https://docs.python.org/es/3/library/functions.html#max)

## 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'])

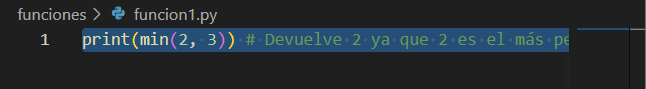
min([])

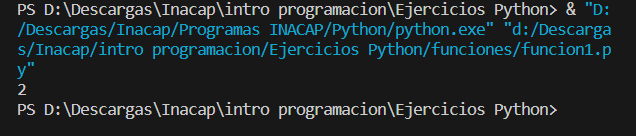
### Valor devuelto

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.

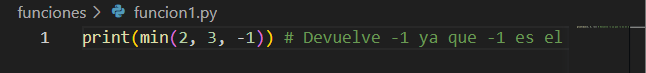
### Código de ejemplo:

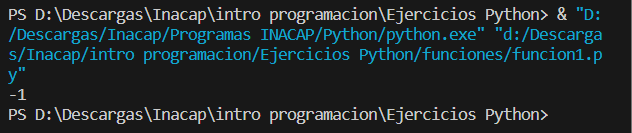
print(min(2, 3)) # Devuelve 2 ya que 2 es el más pequeño de los dos valores





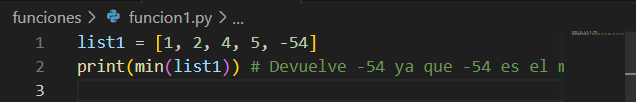
print(min(2, 3, -1)) # Devuelve -1 ya que -1 es el más pequeño de los dos valores

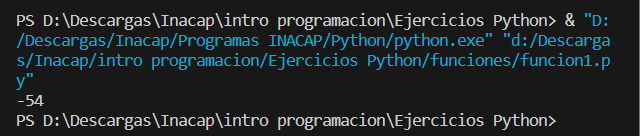




list1 = [1, 2, 4, 5, -54]

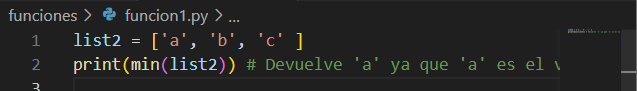
print(min(list1)) # Devuelve -54 ya que -54 es el más pequeño de los valores en la lista.

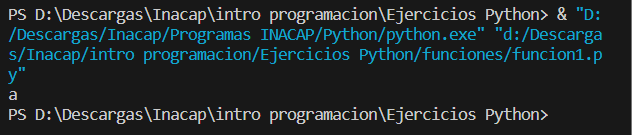




list2 = ['a', 'b', 'c' ]

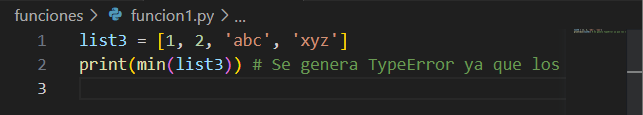
print(min(list2)) # Devuelve 'a' ya que 'a' es el valor más pequeño de la lissta en orden alfabético

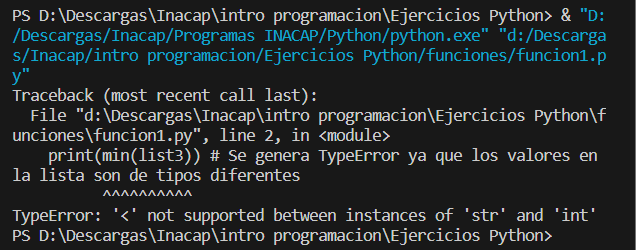




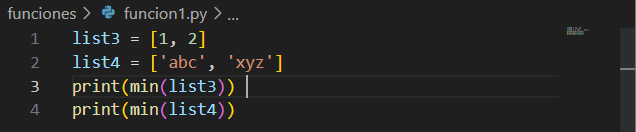
list3 = [1, 2, 'abc', 'xyz']

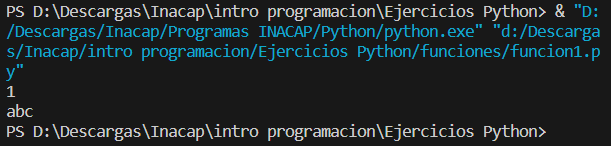
print(min(list3)) # Se genera TypeError ya que los valores en la lista son de tipos diferentes





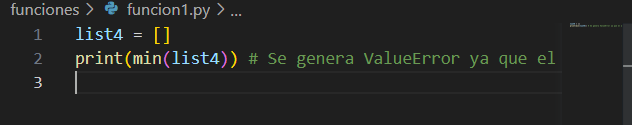
#Soluciona el TypeError mencionado anterioremente antes de continuar con el siguiente paso

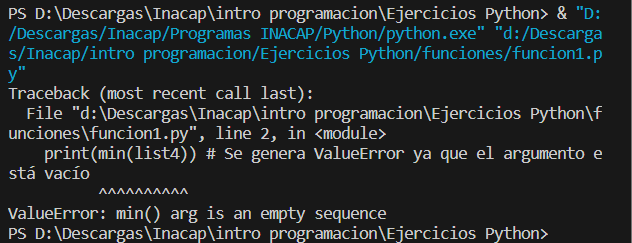




list4 = []

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





[Documentación oficial](https://docs.python.org/es/3/library/functions.html#min)

## 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.

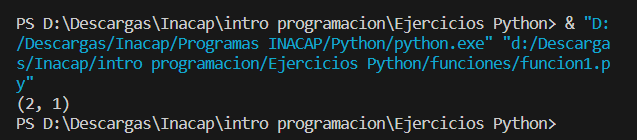
### Valor devuelto

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) pero puede ser 1 menos que eso.

### Código de ejemplo:

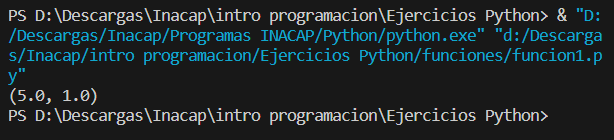
print(divmod(5,2)) # muestra (2,1)





print(divmod(13.5,2.5)) # muestra (5.0, 1.0)

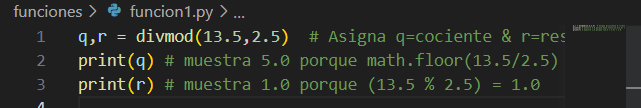


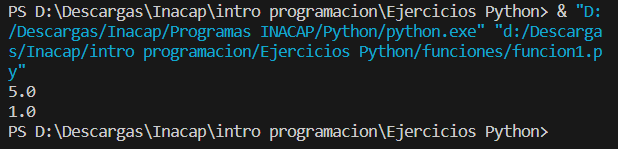


q,r = divmod(13.5,2.5) # Asigna q=cociente & r=resto

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

print(r) # muestra 1.0 porque (13.5 % 2.5) = 1.0





[Documentación oficial](https://docs.python.org/es/3/library/functions.html#divmod)

## Función Hex(x)

hex(x) es una función incorporada en Python 3 para convertir un número entero en una cadena hexadecimal en minúscula con el prefijo "0x"

### Argumentos

Esta función toma un argumento, x, que debería ser de tipo entero.

### Valor devuelto

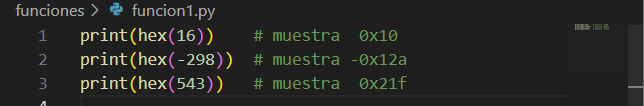
Esta función devuelve una cadena hexadecimal en minúscula con el prefijo "0x".

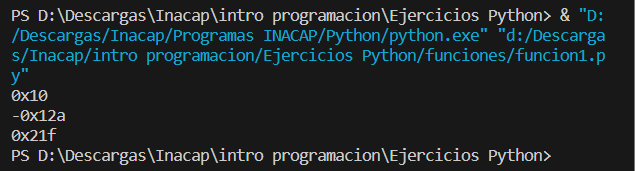
### Código de ejemplo:

print(hex(16)) # muestra 0x10

print(hex(-298)) # muestra -0x12a

print(hex(543)) # muestra 0x21f





[Documentación oficial](https://docs.python.org/es/3/library/functions.html#hex)

## Función len()

len() es una función incorporada en Python 3. Este método devuelve la longitud (el número de elementos) de un objeto. Toma un argumento x.

### Argumentos

Esta función toma un argumento, x. Este argumento puede ser una secuencia (como un string, bytes, tupla, lista o rango) o una colección (como un diccionario, conjunto o conjunto congelado).

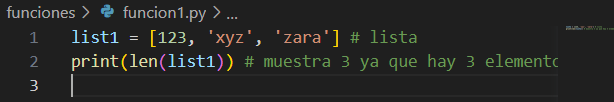
### Valor devuelto

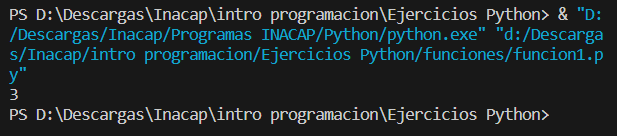
Esta función devuelve el número de elementos del argumento que se pasa a la función len().

### Código de ejemplo:

list1 = [123, 'xyz', 'zara'] # lista

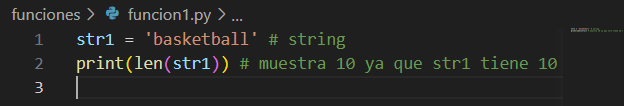
print(len(list1)) # muestra 3 ya que hay 3 elementos en list1

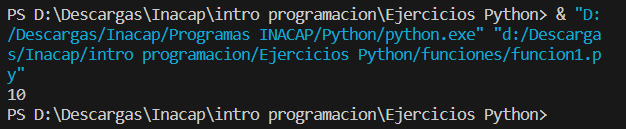




str1 = 'basketball' # string

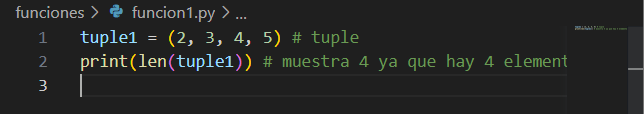
print(len(str1)) # muestra 10 ya que str1 tiene 10 carácteres

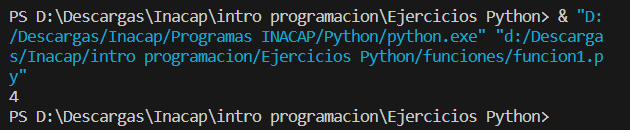




tuple1 = (2, 3, 4, 5) # tuple

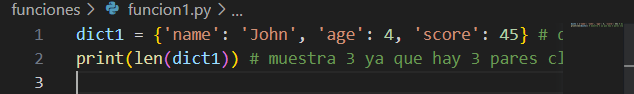
print(len(tuple1)) # muestra 4 ya que hay 4 elementos en tuple1

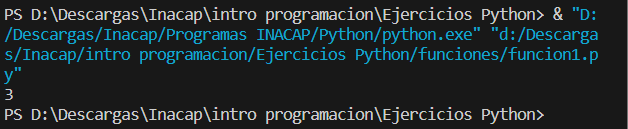




dict1 = {'name': 'John', 'age': 4, 'score': 45} # dictionary

print(len(dict1)) # muestra 3 ya que hay 3 pares clave/valor en dict1





[Documentación oficial](https://docs.python.org/es/3/library/functions.html#len)

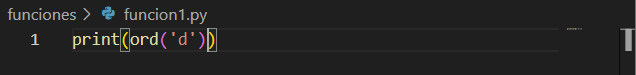
## **Función Ord**

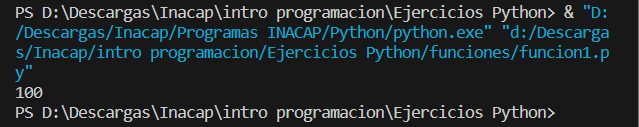
ord() es una función incorporada en Python 3, para convertir la cadena que representa un carácter Unicode en un entero que representa el código Unicode del carácter.

#### **Ejemplos:**

>>> ord('d')

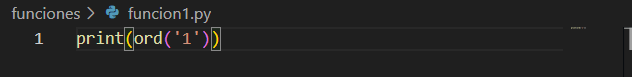
100

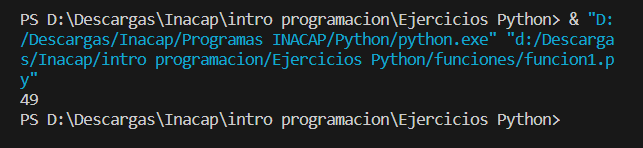




>>> ord('1')

49





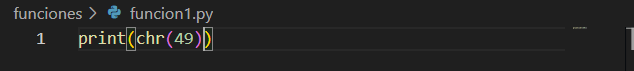
## Función chr

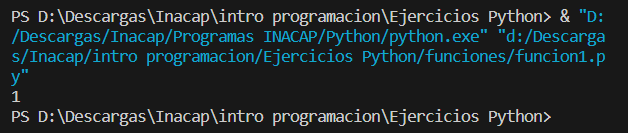
chr() es una función incorporada en Python 3, para convertir el número entero que representa el código Unicode en una cadena que representa un carácter correspondiente.

#### **Ejemplos:**

>>> chr(49)

'1'





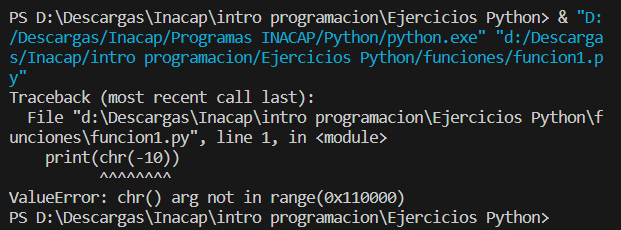
Hay que tener en cuenta que, si el valor entero pasado a chr() está fuera del rango, entonces se generará un ValueError.

>>> chr(-10)

'Traceback (most recent call last):

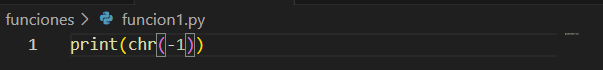
File "<pyshell#24>", line 1, in <module>

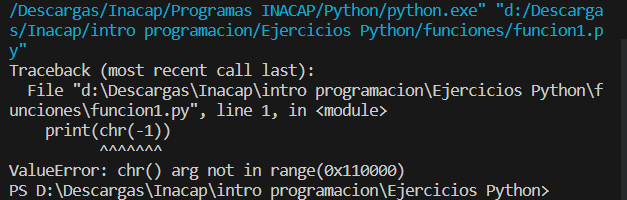




chr(-1)

ValueError: chr() arg not in range(0x110000)'





## Funciones input()

Muchas veces, en un programa necesitamos alguna información del usuario. Recibir información del usuario hace que el programa se sienta interactivo. En Python 3, para recibir datos del usuario tenemos la función input(). Si se llama a la función input, el flujo del programa se detendrá hasta que el usuario haya introducido la información y haya finalizado la introducción de información con la tecla de retorno. Veamos algunos ejemplos:

1. Cuando solo queremos recibir información:

inp = input()

2. Para mostrar en la línea de comandos un mensaje:

promptwithmessage = input(’‘)

3. Cuando queremos tomar un número entero como información:

number = int(input('Please enter a number: '))

Sin embargo, las entradas se almacenan de forma predeterminada como un string. El uso de la función str() deja en claro al lector de código que la entrada será una "cadena". Es una buena práctica mencionar de antemano qué tipo de información se tomará.

## Como llamar una función en Python

Una sentencia de definición de función no ejecuta la función. La ejecución (llamada) de una función se realiza utilizando el nombre de la función seguido de paréntesis que encierran los argumentos requeridos (si los hay).

>>> def di\_hola():

... print('Hello')

...

>>> di\_hola()

Hello

La ejecución de una función introduce una nueva tabla de símbolos utilizada para las variables locales de la función. Más precisamente, todas las asignaciones de variables en una función almacenan el valor en la tabla de símbolos local; mientras que las referencias de variables primero buscan en la tabla de símbolos local, luego en las tablas de símbolos locales de las funciones adjuntas, luego en la tabla de símbolos global y finalmente en la tabla de nombres incorporados. Por lo tanto, a las variables globales no se les puede asignar directamente un valor dentro de una función (a menos que se mencionen en una declaración global), aunque se puede hacer referencia a ellas.

>>> a = 1

>>> b = 10

>>> def fn():

... print(a) # la variable local a no esta asignada, no esta incluida en la función, es una variable global

... b = 20 # la variable local b está asignada en la tabla de símbolos locales para la función.

... print(b) # la variable local b esta referenciada.

...

>>> fn()

1

20

>>> b # la variable global b no se ha modificado por la llamada a la función.

10

Los parámetros actuales (argumentos) para una llamada de función se introducen en la tabla de símbolos local de la función llamada cuando se llama; por lo tanto, los argumentos se pasan usando la llamada por valor (donde el valor es siempre una referencia de objeto, no el valor del objeto). Cuando una función llama a otra función, se crea una nueva tabla de símbolos local para esa llamada.

>>> def saluda(s):

... s = "Hello " + s # s en la tabla local de símbolos es reasignada

... print(s)

...

>>> persona = "Bob"

>>> saluda(persona)

Hello Bob

>>> person # la variable persona utilizada en la llamada continua vinculada al objeto original, 'Bob'.

'Bob'

Los argumentos utilizados para llamar a una función no pueden ser reasignados por la función, pero los argumentos que hacen referencia a objetos mutables pueden cambiar sus valores:

>>> def fn(arg):

... arg.append(1)

...

>>> a = [1, 2, 3]

>>> fn(a)

>>> a

[1, 2, 3, 1]