

«Talento Tech»

# Inteligencia Artificial

Clase 04





## Clase N° 4 | FUNCIONES

### Temario:

- Funciones

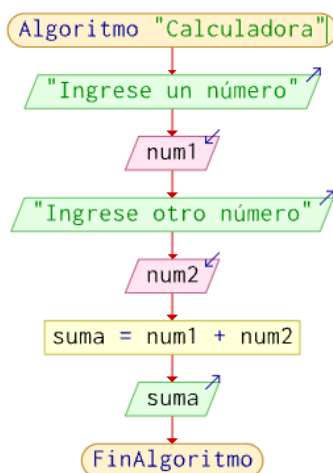


## Funciones

Supongamos que queremos hacer una calculadora, vamos a tener que programar al menos las cuatro funciones básicas: Sumar, restar, multiplicar y dividir.

Ahora bien, si lo hacemos con lo que sabemos hasta ahora, ¿Cómo lo resolverías?

Seguramente encontramos muchas formas de crearla. Pero seguramente todas van a necesitar de un código largo y repetitivo, para cada vez que queramos realizar una operación. Veamos un ejemplo, con las instrucciones más simples que vimos hasta el momento:



En este caso, tendríamos que preguntar si quiere volver a sumar, y volver a escribir todo ese código. Un poco tedioso, ¿No?

Existe una forma que veremos más adelante de poder simplificar esto y poder escribir una sola vez las instrucciones, para luego repetirlas las cantidades de veces que las necesitemos.

Entonces una función es un bloque de código reutilizable que debe realizar una tarea específica, permitiéndonos organizar el código en piezas más manejables y reutilizables, facilitando la lectura y el mantenimiento de un programa. En python tenemos varias formas de crear una función, nosotros vamos a centrarnos en las que serán

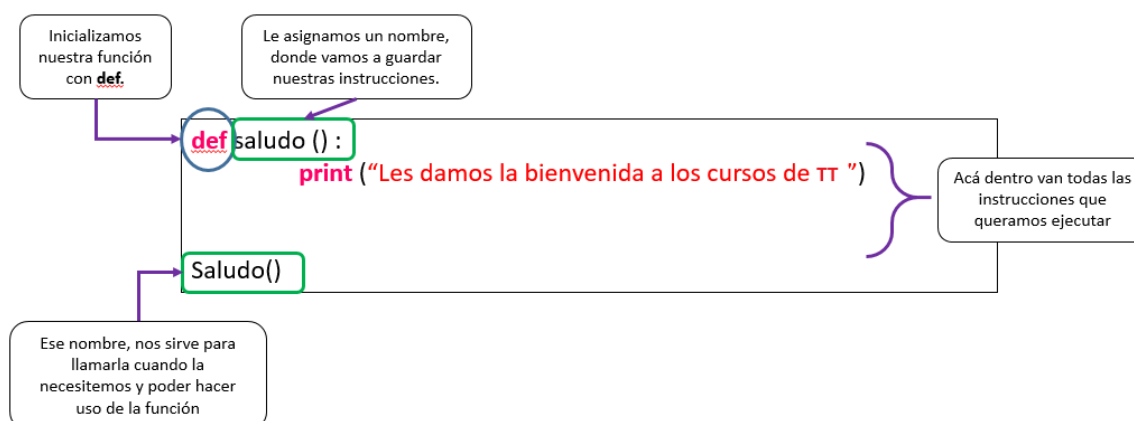
definidas por el usuario (también usaremos los métodos de las clases, como la función `append` de las listas), pero para ir de a poco las vamos a dividir en dos: **funciones simples** y **funciones con parámetros**.

## Funciones simples con Python

Como en todos los lenguajes de programación, debemos respetar una estructura al momento de crear nuestra función

```
def saludo():
    print("Les damos la bienvenida a los cursos de Talento Tech")
```

Veamos parte por parte:



Una vez que declaramos nuestra función con **def**, le debemos asignar un nombre ¿Cómo hacemos con las variables ¿Te acordás?. A ese nombre, por el momento le vamos a dejar unos paréntesis vacíos (¡Por el momento!),

No podemos olvidarnos de nuestros famosos dos puntos, los que nos permiten dar inicio a nuestra función. A partir de ahí, solo nos queda completar con todas las instrucciones que necesitemos. Por ejemplo, podemos hacer una para cada operación de nuestra calculadora.

Hasta el momento solo las declaramos. Es decir, todavía no funcionan, y no veremos su resultado, a no ser que las llamemos. Estos llamados pueden hacerse en cualquier parte de nuestro código, cuando las necesitemos, y la cantidad de veces que queramos.

## Funciones con parámetros

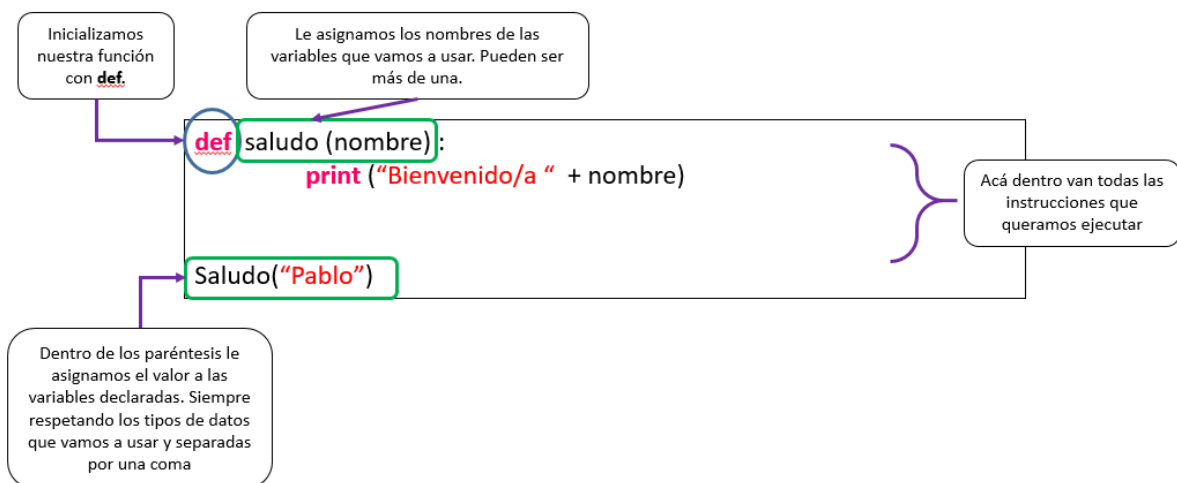
Hasta el momento ya vimos funciones y cómo trabajan. Te proponemos que sigamos con el ejemplo de la calculadora, pero que uses el mismo operador para dos cuentas diferentes. Es decir, primero hacemos  $2 + 2$ , y luego,  $10 + 10$ . En ambos casos queremos que sean casos separados y ver sus resultados. ¿Qué queremos decir con casos separados? Que podamos llamar a cada operación en diferentes momentos de un programa.

### ¿Cómo lo resolverías con funciones?

Seguramente se encontraron con que había que armar dos funciones diferentes, y eso sucede porque ya teníamos algunos argumentos declarados dentro de la función, y si `num1` vale 2, lo hará siempre que se lo llame..

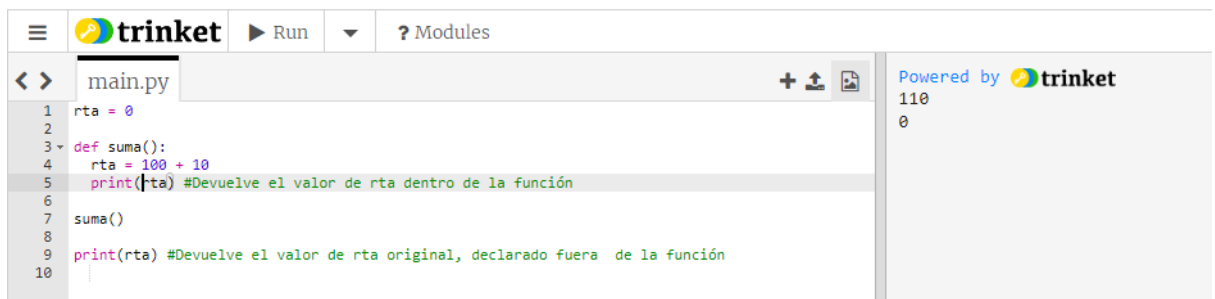
Ahora, ¿Tenemos un modo de primero crear nuestras operaciones y luego declarar los valores de las variables? La respuesta es **SÍ**, y eso lo vamos a lograr utilizando los **PARÁMETROS**. Los cuales vamos a ubicar en esos paréntesis que dejamos vacíos hace un ratito.

Veamos cómo quedaría nuestra estructura:



Ahora que entendemos cómo trabajan los parámetros, solo nos queda entender una pequeña diferencia más. Los parámetros los podemos encontrar por **valor** o por **referencia**.

- Paso por valor:** Cuando se utilizan estos tipos de argumentos, lo que realmente se busca es copiar el valor de las variables en los respectivos parámetros. Cualquier modificación del valor del parámetro, no afecta a la variable externa correspondiente. Este modo es que utiliza Python por defecto, ya que los datos se declaran **inmutables**, es decir que **no pueden cambiar**.  
 Veamos este mini ejemplo:



```

1 rta = 0
2
3 def suma():
4     rta = 100 + 10
5     print(rta) #Devuelve el valor de rta dentro de la función
6
7 suma()
8
9 print(rta) #Devuelve el valor de rta original, declarado fuera de la función
10
    
```

Output: 110, 0

En este caso tenemos una variable declarada por fuera de la función que es “**rta**”. La cual cambia su valor dentro de la función, sin inconvenientes. ¿Pero qué pasa cuando es llamada por fuera de la función? En ese caso, volvió a retomar su valor original, el cual era cero

- Paso por referencia:** En este caso, lo que realmente se busca hacer es copiar en los parámetros la dirección de memoria de las variables que se usan como argumento. Esto implica que realmente se haga referencia al mismo objeto/elemento y cualquier



modificación que se realiza en el parámetro, también afectará a la variable externa correspondiente.

- **Para entender mejor estas referencias hagamos uso de la siguiente metáfora:**

Supongamos que mi hermano me pidió prestado el departamento. En ese caso, yo le paso la dirección, piso y departamento y le doy la llave. Por lo que siempre que quiera hacer uso de mi departamento, tendrá que ir hasta la dirección, al piso, buscar el departamento y usar la llave.



¿Pero qué pasa si me pide un pendrive?

En ese caso, no va a tener que ir hasta donde yo esté para hacer uso del pendrive, sino que se lo puedo dar y que él lo use como quiera. Podría subir y bajar archivos, y al final del día me lo devolvería.

- De la misma forma se manejan los tipos de datos en python. Si el tipo de dato está hecho generalmente para almacenar valores grandes, como puede ser una lista, una tupla o un diccionario, entonces nosotros no le vamos a pasar todos los datos, **sino que le pasamos la dirección en memoria** donde se encuentran los datos. En este caso, los datos serían la dirección, el departamento, por lo que si tira una pared a mi me lo devuelve sin una pared., En caso contrario, si le pasamos un parámetro de tipo primitivo como puede ser un float, entero o string, le pasamos el dato, y no su dirección en memoria. En este caso, los datos serían el pendrive. y todo cambio se le realizará a la variable pero no a su espacio en memoria, por lo que su valor

perdura después de ser trabajado.

Veámoslo con un ejemplo:

```
lista = ['a', 'b', 'c', 'd']
texto = 'hola'

def multi(valor, referencia):
    valor*=2
    referencia*=2
    print(f'durante el valor es "{valor}" y la referencia es "{referencia}"')

print(f'Antes el valor era "{texto}" y la referencia era "{lista}"')

multi(lista, texto)

print(f'Después el valor vale "{lista}" y la referencia "{texto}"')
```

En las primeras líneas nosotros creamos dos variables, una de **tipo primitivo** y la otra es una **lista**.

Por lo general una variable de tipo primitivo (enteros, flotantes, booleanos y cadenas o strings) te permite trabajar con su valor de una forma “superficial”. Mientras que existen otros tipos de datos que tienen la particularidad de ser creados para almacenar gran cantidad de elementos iguales o distintos, como es el caso de la variable lista.





Veamos los resultados de nuestro ejemplo:

```
... Antes el valor era "Hola" y la referencia era "['a', 'b', 'c', 'd']"
    durante el valor es "['a', 'b', 'c', 'd', 'a', 'b', 'c', 'd']" y la referencia es "HolaHola"
    Después el valor vale "['a', 'b', 'c', 'd', 'a', 'b', 'c', 'd']" y la referencia 'Hola'
```

Creamos dos variables, la variable **lista1** que es una lista que contiene dos strings, y

la variable **texto1** que contiene el mismo tipo de dato. Podemos ver que al multiplicar la variable lista1 por dos, nos devuelve una lista de 4 elementos, repitiendo por duplicado sus datos.

En cambio si multiplico la variable texto1, python nos devuelve un string con el valor "HolaHola".

En conclusión: mientras un primitivo nos devuelve su valor dejando inmutable su espacio en memoria, otros tipos de datos más grandes, nos dan acceso a su espacio en memoria, permitiéndonos el cambio.

## Return

Para terminar con las funciones, vamos a ver una de las instrucciones que más nos puede servir, sin importar el lenguaje que utilicemos, y es el **return**.

Este comando nos va a permitir realizar dos acciones:

1. Devolver uno o varios parámetros, como resultados de la función.
2. Salir de la función y volver al punto del programa en donde se realizó la llamada,

Ambos puntos son realmente importantes, pero debemos tener en cuenta una característica específica del return, y es que al utilizarlo, todo lo que esté luego de este comando, va a ser ignorado por la función. Ya que (Como marcamos en uno de sus puntos) vuelve al punto en

donde fue llamada la función. Algo que nos es realmente útil al momento de ejecutar los programas.

## Diferencias entre return y print

Más allá de dar por terminada la función, el **return** tiene otro objetivo y es el de **devolver un valor**, y mostrarlo en consola.

Pero, ¿No es similar al uso del print?

En realidad no, el print lo vamos a utilizar siempre que queramos mostrar un mensaje por la consola, pero no se produce ninguna otra acción.

En cambio con el return, no solo se va a mostrar el nuevo valor por consola, sino que **también permite guardar ese resultado**.

Veamos un ejemplo:

```
def saludo(nombre):  
    return "Hola " + nombre  
  
bienvenida = saludo("Martina")  
print(bienvenida)
```

¿Cuál creés que sería el resultado?

En este caso, por consola vamos a ver el mensaje **“Hola Martina”**. Pero ¿Cómo llegamos a eso?

En el return tenemos una instrucción que solemos utilizar dentro de un print, sin tener que utilizarla de forma explícita. Ese mensaje se está guardando dentro de la función.

Y para ver más claro cual ese valor guardado, en nuestro ejemplo, lo guardamos en una variable nueva.

Al mostrar esa variable podemos ver de qué modo nuestra función guarda ese valor que generamos.

## Funciones predefinidas

Con Python tenemos muchas funcionalidades que podemos realizar desde VsCode sin definir las directamente como funciones. Es decir, que estas funciones trabajan por detrás de nuestro código visual, lo que permite que nuestro programa sea mucho más dinámico, legible y corto.

Estas funciones ya preestablecidas resuelven situaciones sencillas que pueden surgir en el día a día como puede ser la suma, división, longitud, el módulo, etc. Veamos ahora algunas de las funciones más usadas:

### Funciones de cadena y numéricas que tenemos predefinidas

Algunas funciones las utilizaremos más adelante

| Función              | Utilidad   | Ejemplo   | Resultado       |
|----------------------|--|---|-----------------|
| <code>print()</code> | Imprime en pantalla el argumento.                  | <code>print ("Hola")</code>   | "Hola"          |
| <code>sum()</code>   | Suma el total de una lista de números              | <code>x = [5, 5, 5]</code><br><code>print (sum(x))</code><br><code>print(sum([5, 5, 5]))</code> | 15              |
| <code>len()</code>   | Determina la longitud en caracteres de una cadena. | <code>len("Hola Python")</code>   | 11              |
| <code>range()</code> | Crea una lista como rango de números               | <code>x = range (5)</code><br><code>print (list(x))</code>                                      | [0, 1, 2, 3, 4] |
| <code>str()</code>   | Convierte un valor de tipo numérico a tipo texto   | <code>str(22)</code>  | '22'            |
| <code>int()</code>   | Convierte a valor entero                           | <code>int('22')</code>  | 22              |
| <code>float()</code> | Convierte un valor a decimal o punto flotante      | <code>float('2.22')</code>  | 2.22            |

|         |  |  |  |
|---------|--|--|--|
| max()   | Determina el máximo entre una lista de números     | x = [0, 1, 2]<br>print (max(x))  | 2  |
| min()   | Determina el mínimo entre una lista de números     | x = [0, 1, 2]<br>print (min(x))  | 0  |
| round() | Redondea después de la coma de un decimal          | print (round(1.55))  | 2  |
| type()  | Devuelve el tipo de un elemento                    | x = 9<br>type(x)   | int  |
| input() | Permite la entrada de datos al usuario en Python 3 | y = int(input("Ingrese el número"))<br>print (f'y vale {y}')             | ingreso el valor 5 por consola<br>y vale 5 |
| join()  | Convierte en cadena utilizando una separación      | lista = ['Python', 'es']<br>'-'.join(lista)                              | 'Python-es'                                |
| split() | Convierte una cadena con un separador en una lista | a = ("hola esto sera una lista")<br>lista2 = a.split()<br>print (lista2) | ['hola', 'esto', 'sera', 'una', 'lista']   |
| upper() | Convierte una cadena de texto en mayúscula         | texto = "Estamos aprendiendo Python"<br>texto.upper()                    | "ESTAMOS APRENDIENDO PYTHON"               |
| lower() | Convierte una cadena de texto en minúscula         | texto2 = "Esto Será TODO MinuSCULA"<br>texto2.lower()                    | "esto será todo minuscula"                 |



## Repaso

### Ejercicios con operadores y condicionales

#### Ejercicio 1: Comparación de Números

Escribe una función que pida al usuario dos números y muestre en pantalla cuál es el mayor.

#### Ejercicio 2: Verificación de Paridad

Crea una función que solicite un número al usuario y determine si es par o impar.

#### Ejercicio 3: Categorización de Edad

Desarrolla una función que clasifique la edad ingresada por el usuario en categorías: niño, adolescente, adulto.

#### Ejercicio 4: Validación de Contraseña

Crea una función que solicite al usuario ingresar una contraseña y la valide. La contraseña debe tener al menos 8 caracteres y un carácter especial.

#### Ejercicio 5: Días en un Mes

Desarrolla una función que solicite al usuario el nombre de un mes y muestre la cantidad de días que tiene.

#### Ejercicio 6: Rango de Números

Escribe una función que solicite al usuario un número y determine en qué rango se encuentra: negativo, positivo o cero.



### Ejercicio 7: Calculadora Básica

Crea una función que realice operaciones básicas (suma, resta, multiplicación, división) según la elección del usuario.

### Ejercicio 8: Determinar Números Positivos y Negativos

Desarrolla una función que tome tres números y determine cuántos son positivos, cuántos son negativos y la suma entre ellos.

### Ejercicio 9: Verificación de Año Bisiesto

Crea una función que determine si un año ingresado por el usuario es bisiesto o no

Para que un año sea considerado bisiesto, debe cumplir las siguientes reglas:

- Divisible por 4:** El año debe ser divisible por 4.
- Divisible por 400 pero no por 100:** Sin embargo, si el año es divisible por 100, no es un año bisiesto. A menos que el año sea también divisible por 400. Si es divisible por 400, sí es un año bisiesto. 🤖🤖

### Ejercicio 10: Calcular el IVA.

Escribir una función que calcule el total de una factura tras aplicarle el IVA. La función debe recibir la cantidad sin IVA y el porcentaje de IVA a aplicar, y devolver el total de la factura. Si se invoca la función sin pasarle el porcentaje de IVA, deberá aplicar un 21%.

### Ejercicio 11: Función de Login

Crear una función llamada "Login", que recibe un nombre de usuario y una contraseña y te devuelve Verdadero si el nombre de usuario es "usuario1" y la contraseña es "messiCrack".



Además recibe el número de intentos que se ha intentado hacer login y si no se ha podido hacer login incrementa este valor.

## Ejercicio con bucles

### **Ejercicio 1: Contador Simple**

Escribe una función que utilice un bucle for para imprimir los números del 1 al 5.

### **Ejercicio 2: Suma de Números**

Desarrolla una función que solicite al usuario un número n y calcule la suma de los números del 1 al n.

### **Ejercicio 3: Tabla de Multiplicar**

Crea una función que solicite al usuario un número y muestre su tabla de multiplicar del 1 al 10.

### **Ejercicio 4: Números Pares**

Escribe una función que imprima los números pares del 2 al 10 utilizando un bucle while.

### **Ejercicio 5: Invertir una Cadena**

Crea una función que solicite al usuario ingresar una cadena y la imprima en orden inverso.

### **Ejercicio 6: Contador de Vocales**

Desarrolla una función que cuente el número de vocales en una palabra ingresada por el usuario.

### **Ejercicio 7: Suma de Números con Salto**



Crea una función que calcule la suma de los números del 1 al 20, pero solo considerando aquellos que son múltiplos de 3.

### **Ejercicio 8: Área de figuras.**

Escribir una función que calcule el área de un círculo y otra que calcule el volumen de un cilindro usando la primera función.

### **Ejercicio 9: Mostrando información.**

Escribí una función que reciba dos parámetros. Los dos deben ser strings y los mostraremos por la terminal. Un parámetro debe ser el título de un artículo y el otro un párrafo corto.

### **Ejercicio 10: Convertir unidades**

Escribir dos funciones que permitan calcular:

La cantidad de segundos en un tiempo dado en horas, minutos y segundos.

La cantidad de horas, minutos y segundos de un tiempo dado en segundos.

Escribe un programa principal con un menú donde se pueda elegir la opción de convertir a segundos, convertir a horas, minutos y segundos o salir del programa.

### **Ejercicio 11: Guardando información**

Escribir un programa que pregunte al usuario su nombre, apellido, edad, estudios y trabajo y lo guarde en un diccionario. Después debe mostrar por pantalla un mensaje completando una oración con todos sus datos.





## Ejercicio 12: Contador de palabras



Escribir un programa que reciba una cadena de caracteres y devuelva un diccionario con cada palabra que contiene y su frecuencia. Escribir otra función que reciba el diccionario generado con la función anterior y devuelva una tupla con la palabra más repetida y su frecuencia.

## Desafío N° 4:

1. Crea una función llamada "**marcar\_pares**" que reciba un número entero positivo e imprima los números pares desde 0 hasta ese número. Para los números pares, la función debe imprimir "par" junto con el número, y para los impares, debe imprimir solamente el número.

Te doy unas pistas para ayudarte a resolver este desafío:

### Pistas.

1. **Pista para imprimir números pares:**  ¡Identificando Pares!  
Imagina que estás organizando una fiesta y quieres etiquetar los números pares. Crea una función que recorra todos los números desde 0 hasta el número que introduzcas. Para cada número, usa un condicional para verificar si es par. Si lo es, imprime el número seguido de "par". Si no es par, solo imprime el número.
2. **Pista para imprimir números impares:**  ¡Detección de Impares!  
Piensa en la función como un detective que busca números pares. Dentro del bucle, verifica si el número actual no es par. Si no lo es, simplemente imprime el número. Recuerda, solo necesitas agregar "par" a los números que sean divisibles por 2.



**Buenos Aires**  
*aprende*  
Agencia de Actividades para el Futuro

**BA** Buenos  
Aires  
Ciudad