Clase 09 - Funciones

# **Funciones**

Cuando creamos nuestros propios programas nos damos cuenta de que muchas de las tareas que implementamos se repiten o se presentan de forma similar pero con algunos cambios.

Entonces aparece la necesidad de agrupar este código repetido o similar, a las agrupaciones de código se les denomina funciones las cuales se pueden ejecutar múltiples veces gracias a un nombre único que las identifica.

Para comunicarse con nuestro proceso principal, las funciones pueden recibir y devolver datos manipulados. Un ejemplo de una función que conocemos es len() que nos permite saber la cantidad de elementos de una colección.

Recordemos que a esta función hay que pasarle el elemento del cual queremos saber la longitud y devuelve un valor entero con la longitud, a este valor se le denomina valor de retorno.

**>>> len(“Hola”)**

**4**

## **DEF**

### ¿De qué se trata?

La sentencia def sirve para crear funciones definidas por el usuario. Una definición de función es una sentencia ejecutable.

### Sintaxis para una definición de función

* Nombre: Es el nombre de la función
* Parámetro: Como vimos en la clase 8 hay scripts con argumentos, en las funciones, cuando recibe argumentos, se les denominan parámetros.
* Sentencias: Es el bloque del código.
* Return: Es una sentencia de Python, le indica a la función qué devolver cuando llamamos a la función
* Expresión: Es lo que devuelve la sentencia return.

>>> def NOMBRE(PARÁMETROS):

SENTENCIAS

RETURN [EXPRESIÓN]

### **Definir funciones básicas**

**def saludar():**

**print('Estoy saludando desde la función')**

La llamamos usando**: saludar()**

Estoy saludando desde la función

### Definir funciones más avanzadas



Prueba el código para ver qué pasa 🙂

### **Recomendaciones**

* Utilizar minúsculas
* Las palabras se separan con guiones bajos \_
* Utilizar nombres autoexplicativos
* No usar nombres que no definan lo que hace la función (ejemplo letras simples o palabras sin sentido con lo que haga la función)

### **Variables y funciones**

**def test():**

**variable\_test = 10**

**print(variable\_test)**

**print(variable\_test)**

**NameError: name 'variable\_test' is not defined**

Hay que tener en cuenta que las variables creadas en una función no existen fuera de la misma, de decir son variables locales.

### **Variables y funciones**

Sin embargo, hay que tener cuidado con las variables fuera de las funciones al usarlas en una función, ya que no puede llegar a funcionar como queremos:

**variable\_test = 10**

**def test():**

**variable\_test = 155**

**print(variable\_test)**

**test()**

**>>>155**

El print le da prioridad a la variable dentro de la función antes que a la de afuera.

Usar para slides de solo texto. Si no alcanza, no sobrecargar, usar otra con el mismo título para indicar que continúa el mismo módulo.

# **Retornando valores**

## **Return**

Las funciones pueden comunicarse con el exterior de las mismas, al proceso principal del programa usando la sentencia return. La comunicación con el exterior se hace devolviendo valores.

A continuación, un ejemplo de función usando return:

**def saludar\_con\_nombre(nombre):**

**saludando = print('Hola {}! ¿Cómo estás?'.format(nombre))**

**return saludando**

**saludar\_con\_nombre("Juan")**

**>>>Hola Juan ¿Cómo estás?**

Nota: Por defecto, las funciones retorna el valor None.

Sin embargo hay que tener en cuenta que la función termina al devolver un valor, es decir, lo que escribamos después no se ejecutará:

**def saludar\_con\_nombre(nombre):**

**saludando = print('Hola {}! ¿Cómo estás?'.format(nombre))**

**return saludando**

**print("Hola Mundo")**

**👆** ¡Es similar a un break!

Algo interesante que pasa si devolvemos una colección es que podemos utilizarla directamente desde la función y hacer uso de las funciones internas de las colecciones:

Sin embargo, cada vez que hagamos un print a una función la estaremos llamando, por lo que lo ideal es asignarlo a una variable y trabajarlo desde ahí:

**def lista():**

**return [1,2,3,4,5]**

**print(lista()[1:3])**

**variable = lista()**

**variable[1:4]**

Una característica interesante, es la posibilidad de devolver valores múltiples separados por comas:

**def test():**

**return "Python", 20, [1,2,3]**

**test()**

**('Python', 20, [1, 2, 3])**

# **Enviando valores**

## **Enviando valores a una función**

Vimos como devolver valores y así comunicar una función con el exterior, ahora enviar información desde el exterior a la función.

Para entender los conceptos más fácilmente vamos a trabajar alrededor de un caso de estudio típico: crear una función que sume dos números y retorne uno en su resultado.

## **Enviando valores a una función**

Lo primero será definir una función la cual denominaremos como suma y recibirá 2 números con dos nombres como si fueran dos variables numero1 y numero2, luego retornamos la suma entre ambos números.

**def suma(numero1, numero2):**

**return numero1 + numero2**

Lo que hacemos para indicar que se reciben valores es crear dos variables separadas por una coma. Cuando nosotros llamemos a la función, automáticamente, se le asignarán a estas variables los números que enviemos, siguiendo el mismo orden:

**>>> r = suma(7, 5)**

En este caso 7 será la variable numero1 y 5 será la variable numero2