Funciones

Introducción a funciones

Idea clave

- Las funciones nos permiten reutilizar código.
- Las funciones deben ser definidas una vez y luego pueden ser llamadas a lo largo del programa.
- Palabras clave: **definir**, **llamar**.

¿Qué son las funciones?

Las funciones son bloques de código que nos permiten reutilizar y estructurar nuestro código de una manera más organizada. Al **definir** una función, estamos creando un conjunto de instrucciones que realizan una tarea específica. Luego, podemos **llamar** a esa función en cualquier parte de nuestro programa cada vez que necesitemos ejecutar esa tarea. Esto nos ayuda a evitar la repetición de código en nuestras aplicaciones.

Veamos un ejemplo:

```
# Definimos la función

def saludar():
  print("Hola Mundo")

saludar() # Llamamos a la función
saludar() # Llamamos a la función una segunda vez
saludar() # Llamamos a la función una tercera vez
```

En el código mostrado, primero se define la función `saludar` que muestra el mensaje "Hola Mundo". Luego, llamamos a la función tres veces. Cada vez que llamamos a la función, se ejecuta el código dentro de la función `saludar`, mostrando el mensaje "Hola Mundo".

Ejercicio

Se tiene definida la función `despedir`, pero no se está llamando. Llama a la función despedir para que imprima "Adiós Mundo" en la consola.

```
def despedir():
  print("Adiós Mundo")

# Escribe tu código aquí




# Fin
```

Creando una función

Ideas clave

- Las funciones nos permiten reutilizar código
- Existe más de una forma de crear una función
- Por ahora las crearemos utilizando la palabra clave `def`
- Una función primero debe ser definida para que luego pueda ser llamada

Cómo crear una función en Python

Para crear una función en python utilizamos la palabra clave `def`. Podemos recordarlo como **def**inir una función. La palabra clave `def` va seguida del nombre de la función, un paréntesis y dos puntos.

Por ejemplo, para crear una función llamada "saludar" que muestre "Hola Mundo", escribimos:

```
def saludar():
  print("Hola Mundo");
```

Tip: es importante destacar que existen varias formas de definir funciones en Python, incluyendo algunas más modernas. Sin embargo, en este momento nos centraremos en los conceptos básicos. Posteriormente, exploraremos las diferentes formas de crear funciones, sus diferencias, ventajas y cómo transformar una forma en otra.

Ejercicio

En el siguiente código se está llamando a la función `saludar`, pero no se ha definido. Define la función `saludar` para que imprima "Hola Mundo" en la consola. Respeta las mayúsculas, minúsculas y espacios.

```
# Escribe tu código aquí


# Fin

saludar()
saludar()
saludar()
```

Parámetros

Ideas clave

- Una función puede depender de parámetros para llevar a cabo su tarea.
- Utilizando parámetros, podemos hacer que una función se adapte a diferentes situaciones.

Las funciones como un robot

Consideremos estas dos opciones.

- Tenemos un robot que saluda a cada persona con un "Hola".
- Tenemos un robot que saluda a cada persona con un "Hola" seguido de su nombre, pero para ello necesita saber el nombre de la persona.

Podemos imaginar una función como un robot que realiza una tarea. El primer caso sería una función que muestre el mensaje "Hola" cada vez que se llama. En el segundo caso, es una función que muestra un mensaje personalizado, pero para ello necesita como entrada el nombre de la persona.

¿Qué son los parámetros?

Los parámetros son los valores de los que depende una función para realizar una tarea. Depender de valores hace que una función sea más versátil y adaptable a diferentes situaciones.

Veamos un ejemplo en código de una función que salude a una persona con su nombre:

```
def saludar(nombre):
    print("Bienvenido, " + nombre)

saludar("Alex")  # Bienvenido, Alex
saludar("Juan")  # Bienvenido, Juan
saludar("Pepe")  # Bienvenido, Pepe
```

En el ejemplo mostrado, la función saludar depende del parámetro nombre, que se utiliza para personalizar el mensaje de bienvenida. Al llamar a la función con diferentes nombres, cada llamada produce un mensaje único y personalizado. Esto ilustra cómo los parámetros permiten que una función sea más versátil y adaptable a diversas situaciones.

Ejercicio

Se ha definido una función llamada mostrar_mensaje_secreto. Esta función depende de un parámetro llamado mensaje. Si el mensaje es "secreto", mostrará "Mensaje correcto". Sin embargo, si el mensaje es cualquier otro, la función finalizará, mostrando "Término de la función".

Para resolver el ejercicio debes llamar a la función dos veces, primero, con el mensaje "secreto" y luego con el mensaje "no secreto" para verificar que funciona correctamente.

```
def mostrar_mensaje_secreto(mensaje):
    if mensaje == "secreto":
        print("Mensaje correcto")
    print("Término de la función")

# Escribe tu código aquí



# Fin
```

Múltiples parámetros

Ideas clave

- Una función puede depender de **múltiples** parámetros para llevar a cabo su tarea.

Cómo trabajar con múltiples parámetros

Podemos especificar más de un parámetro en una función separándolos por comas. Por ejemplo, para crear una función que sume dos números, escribiremos:

```
def sumar(a, b):
  print(a + b)



sumar(5, 3) # 8
sumar(10, 20); # 30
```

Ejercicio

Crea la función `restar` que dependa de dos parámetros y que imprima la resta de ambos.

```
# Escribe tu código aquí



# Fin

restar(10, 5)
restar(3, 5)
restar(9, 4)
```

Parámetros y argumentos

Ideas clave

- Una función puede depender de múltiples parámetros para llevar a cabo su tarea.
- Los valores que uno pasa cuando llama a una función formalmente reciben el nombre de argumentos.
- Agregar más parámetros a una función es tan sencillo como separarlos por comas.
- Los argumentos se pasan en el mismo orden que los parámetros de la función.

¿Cómo trabajar con múltiples parámetros?

Así como una función puede depender de uno o dos parámetros, también puede depender de tres o más parámetros.

Por ejemplo, considera la función sumar(a, b, c) que suma tres números y los muestra en la consola:

```
def sumar(a, b, c):
  print(a + b + c)

sumar(5, 3, 2) # 10
```

Los argumentos se pasan en el mismo orden que los parámetros de la función.

Parámetros vs. Argumentos

Existen dos conceptos parecidos pero distintos que es importante tener en cuenta al trabajar con funciones:

Parámetros: Son los nombres que usas en una función para recibir datos. Por ejemplo, en sumar(a, b, c), a, b y c son parámetros.

Argumentos: Son los valores que le das a la función cuando la llamas. Por ejemplo, en sumar(5, 3, 2), 5, 3 y 2 son argumentos.

Ocasionalmente, se usa el término argumento para referirse a ambos conceptos, pero es importante tener en cuenta la diferencia al leer mensajes de error o documentación.

Ejercicio
Crea la función restar que dependa de tres parámetros a, b y c. Esta función al ser llamada debe mostrar el resultado de la resta de los tres valores ingresados.

```
# Escribe tu código aquí


# Fin
restar(10, 5, 3)
restar(3, 5, 2)
restar(9, 4, 1)
```

Valores por defecto

En python, una función puede tener valores por defecto en sus parámetros. Esto significa que si al llamar a la función no se le pasa un valor para un parámetro, este tomará el valor por defecto que se le haya asignado.

Veamos un ejemplo:

```
def saludar(nombre="Mundo"):
  print("Hola ", nombre)

saludar("Juan") # Hola Juan
saludar() # Hola Mundo
```

Puede haber mas de un parámetro con valor por defecto.

```
def saludar(nombre="Mundo", saludo="Hola"):
  print(saludo, nombre)

saludar("Juan") # Hola Juan
saludar() # Hola Mundo
```

Si hay parámetros sin valor por defecto, estos deben ir al principio y los que tienen valor por defecto al final.

```
def saludar(saludo, nombre="Mundo"):
  print(saludo, nombre)

saludar("Hola", "Juan") # Hola Juan
saludar() # Error  saludar() missing 1 required positional argument: 'saludo'
```

Ejercicio

Crea la función suma que reciba tres parámetros y muestre la suma de estos. Si no se le pasa un valor a alguno de los parámetros, este debe tomar el valor por defecto de 0.

Ejemplo:

suma(5, 3) # Es igual a suma(5, 3) # 8
suma(10) # Es igual a suma(10, 0) # 10
suma() # Es igual a suma(0, 0) # 0

```
# Escribe tu código aquí



# Fin
suma(10, 5, 2)
suma(3)
suma(9, 4)
```

Retornar vs Mostrar

Ideas clave

- Las funciones pueden devolver un valor.
- Retornar un valor no es lo mismo que mostrar un valor.
- En algunos ejercicios te pediremos que muestres un valor y en otros que devuelvas un valor.

Mostrar o devolver

Los valores que muestra una función no necesariamente son los mismos que devuelve.

```
def sumar(a, b):
    print("La suma es 8")
    return a + b

sumar(2, 2) # La suma es 8
```

Aquí vemos que la función `sumar` muestra un mensaje, pero el valor que devuelve es la suma de los parámetros `a` y `b`. Es decir, podemos hacer que la función muestre un mensaje y devuelva un valor diferente.

Este ejemplo es importante para entender que **mostrar** un valor y **devolver** un valor no es lo mismo y no están relacionados directamente.

En algunos ejercicios te pediremos que muestres un valor y en otros que devuelvas un valor.

El siguiente ejercicio te ayudará a entender la diferencia entre mostrar y devolver valores.

Ejercicio
Crea la función `sumar_y_duplicar` que dependa de dos parámetros, `a` y `b`, y que **devuelva** la suma de ambos multiplicada por 2. Adicionalmente, antes de retornar el valor, la función debe mostrar el valor de la suma de la siguiente forma: "La suma es X", donde X es el resultado de la suma.

Ejemplo: sumar_y_duplicar(2, 2) # La suma es 4 sumar_y_duplicar(3, 3) # La suma es 6

```
# Escribe tu código aquí




# Fin
print(sumar_y_duplicar(2, 2))
print(sumar_y_duplicar(3, 3))
```

Retorno
Ideas clave

- Retornar un valor no es lo mismo que mostrar un valor.
- Si piden mostrar un valor utilizaremos `print`. Cuando nos piden retornar un valor, debemos utilizar `return`.
- Una función puede tener un retorno utilizando la palabra `return` seguida de una expresión. Ejemplo: `return a + b`.
- Si el return no se especifica, la función retornará `undefined`.

Las funciones como cajas negras

Las funciones son como cajas que guardan instrucciones que tienen puntos de entrada donde podemos enviarle datos, y un punto de salida donde puede devolver datos. Los puntos de entrada se llaman **parámetros** y el punto de salida se llama **retorno**.

Creando una función que retorne un valor

Veamos un ejemplo:

```
def sumar(a, b):
  return a + b
```

```
print(sumar(5, 3)) # 8
print(sumar(10, 20)) # 30
```

En este ejemplo, la función sumar retornará la suma de los parámetros a y b.

El valor que retorna la función se puede guardar en una variable para utilizarlo posteriormente.

```
resultado1 = sumar(5, 3)
resultado2 = sumar(resultado1, 20)
print(resultado2) # 28
```

Por ahora, más que entender los detalles, lo importante es entender los conceptos de **parámetros** y **retorno**, ya que los utilizaremos en ejercicios futuros. La mayoría de los ejercicios consistirá en crear una función que reciba parámetros y retorne un valor específico de acuerdo a las instrucciones.

Ejercicio
Crea la función multiplicar que dependa de dos parámetros y que retorne la multiplicación de ambos.

```
# Escribe tu código aquí



# Fin


print(multiplicar(5,3))
print(multiplicar(10,20))
print(multiplicar(7,8))
```

Sin retorno
Ideas clave

- Retornar un valor no es lo mismo que mostrar un valor.
- Si se pide mostrar un valor debemos utilizar print.
- Si se pide retornar un valor, debemos utilizar return.
- Si el return no se especifica, la función retornará None.

Sin retorno

En Python, cuando una función no tiene explícitamente un return, la función retornará None.

```
def sumar():
    2 + 2  # Realiza una operación pero no devuelve nada

print(sumar())  # None
```

Comparemos el código anterior con uno que sí tiene un return.

```
def sumar():
    return 2 + 2  # Devuelve el resultado de la operación

print(sumar())  # 4
```

¿Por qué una función retorna o no retorna algo?

Estas son decisiones que se toman al crear una función. Tiene sentido que una función no retorne algún valor si su propósito es simplemente realizar una acción, como mostrar un mensaje en pantalla, mientras que las funciones que realizan cálculos o procesos generalmente retornan un valor para que pueda ser utilizado en otro lugar del programa.

Ejercicio
El siguiente código es una función que convierte grados Celsius a Fahrenheit, sin embargo, tiene un problema que debes corregir. Utiliza lo aprendido en esta lección para corregirlo.

```
# Escribe tu código aquí
def a_fahrenheit(celsius):
  celsius * 9 / 5 + 32  # Agregar return para devolver el
resultado

# Fin

r1 = a_fahrenheit(0)
r2 = a_fahrenheit(100)
print("El agua se congela a " + str(r1) + " grados Fahrenheit")
print("El agua hierve a " + str(r2) + " grados Fahrenheit")
```

Aplica lo aprendido.

Ejercicio
Marco Polo es un juego de niños que consiste en que un jugador grita "Marco" y los demás jugadores responden "Polo". El jugador que grita "Marco" debe tratar de encontrar a los demás jugadores usando solo el sonido de sus voces.

Crea la función `marcoPolo` que reciba un texto. Si el texto es "Marco" la función debe retornar "Polo".

Pistas

- Puedes usar el operador == para comparar si dos textos son iguales.
- Respeta mayúsculas y minúsculas.

```
# Escribe tu código aquí



# Fin

print(marcoPolo('Marco'))
```