Buenos Aires aprende
Agencia de Habilidades para el Futuro

Talento Tech

Iniciación con Python

Clase 11 - “Funciones”

Buenos Aires Ciudad

¡Les damos la bienvenida!

 Vamos a comenzar a grabar la clase

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Buenos Aires Ciudad

Talento Tech

Clase 10.

Diccionarios

1. Diccionarios: uso y métodos esenciales.
2. Uso de diccionarios como medio de almacenamiento temporal de datos.

Clase 11.

Funciones

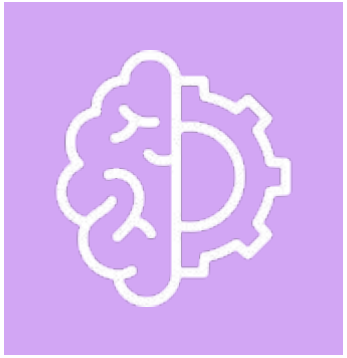
1. Funciones en Python: definición y uso.
2. Argumentos y retorno de valores.
3. Funciones que llaman a funciones.

Clase 12.

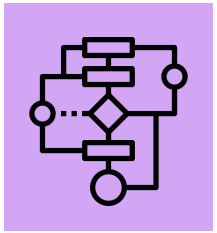
Ruta de avance

1. Creamos las funciones necesarias para el Proyecto Integrador

Pero antes...



¡Resolvamos los “**Ejercicios prácticos**” de la clase anterior!



Funciones

¿Qué es una función en Python?

En programación una función es un bloque reutilizable de código.

Utilizar funciones tiene muchas ventajas. Entre otras:

- Ahorro de tiempo.
- Organización del código.
- Mantenimiento más fácil.
- Evitan la repetición.
- Mejoran la claridad del código.

Definición de una función

Para definir una función en Python usamos la palabra clave def seguida del nombre de la función y un par de paréntesis.

Ejemplo:

```
# La función se define con def y comienza luego de los ":"
def saludar():
    print("Hola, ¡bienvenido!")
```

Llamando a una función

Para ejecutar una función, simplemente debemos “llamarla” (invocarla) por su nombre. Es importante recordar que la función debe ser definida antes de invocarla por primera vez.

Ejemplo:

```
# La función se define con def y comienza luego de los ":"
def saludar():
    print("Hola, ¡bienvenido!")

saludar() #  <-- Aquí invocamos a la función "saludar()"
```

¿Qué son los argumentos?

Los **argumentos** y **parámetros** permiten pasar información a las funciones. En este ejemplo, a y b son **parámetros** de la función sumar. Cuando llamás a la función, le pasás dos valores (**argumentos**) que ocuparán esos lugares:

Ejemplo:

```
def sumar(a, b):
    resultado = a + b
    print("El resultado es:", resultado)

sumar(5, 3)
```

Tipos de argumentos

El uso de argumentos permite que las funciones sean mucho más dinámicas y reutilizables. Los argumentos no tienen que limitarse a números. Podés pasarle cadenas, listas, diccionarios, o cualquier tipo de dato que Python soporte.

Ejemplo con cadenas:

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

saludar("Ana")      # Hola Ana
saludar("Carlos")   # Hola Carlos
```

Argumentos por defecto

Los argumentos pueden tener valores “por defecto”, lo que hace más versátiles a las funciones. Esto se consigue igualando los parámetros al valor que queramos usar por defecto:

Ejemplo:

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

saludar()  # Usa el valor por defecto
saludar("Lucía")  # Sobrescribe el valor por defecto
```

Argumentos nombrados

Podés pasar los argumentos especificando el nombre del parámetro. Esto te permite alterar el orden y hace que el código sea más claro.

Ejemplo:

```
def sumar(a, b):
   resultado = a + b
   print("El resultado es:", resultado)

sumar(b=7, a=2)
# Salida: El resultado es: 9
```

Funciones que devuelven valores

Las funciones pueden retornar valores, esto les da a las funciones un nivel adicional de utilidad y flexibilidad. Esto es posible utilizando la orden **return**.

Al utilizar return, le estás diciendo a la función que una vez que haya hecho su trabajo, debe "regresar" un valor al código que la llamó, para que se pueda hacer algo más con él.

Por ejemplo, si tenés una función que puede sumar dos números y muestra el resultado con print(), la función simplemente mostraría el resultado, pero no podrías hacer nada más con él. Pero si usás return, podés usar ese resultado de manera más amplia en el resto de tu programa. Es una forma de mantener el flujo de información y acciones dentro del código.

Uso de return

Veamos un ejemplo básico de una función que devuelve un valor.

Ejemplo de return:

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

total = sumar(5, 3)
print("El total es:", total)
```

Funciones que llaman a otras

Dentro de una función es posible invocar a otra función. Esto es algo súper útil porque permite dividir tareas complejas en partes más pequeñas y fáciles de manejar, además de reutilizar código. En lugar de escribir todo de nuevo, podés aprovechar funciones ya definidas para resolver partes de un problema mayor.

```python
# Función para calcular las ventas de un producto
def calcular_ventas_producto1 ():
    ventas = 10  # Ejemplo de ventas
    return ventas

# Función para calcular las ventas de otro producto
def calcular_ventas_producto2 ():
    ventas = 15  # Ejemplo de ventas
    return ventas

# Función que llama a las anteriores para calcular el total
def calcular_ventas_totales ():
    total = calcular_ventas_producto1 () + calcular_ventas_producto2 ()
    return total

# Llamamos a la función principal
ventas_totales = calcular_ventas_totales ()
print("Las ventas totales son:" , ventas_totales
```

Alcance de las variables

Con “alcance de las variables” nos referimos a dónde y cómo podés usar una variable dentro de tu código.
En Python, las variables que se crean dentro de una función tienen un comportamiento especial: solo existen mientras esa función se está ejecutando. Eso quiere decir que, si creás una variable dentro de una función, no podés usarla afuera de la misma, y una vez que la función termina, esa variable desaparece.

Variable “local”:

```
def saludar():
    mensaje = "Hola, ¿cómo estás?"
    print(mensaje)

saludar()
print(mensaje)   # Esto va a dar un error
```

Alcance de las variables

Si una variable se crea fuera de una función, esa variable puede usarse tanto fuera como dentro de la función. Se dice que su alcance es “global”.

Variable “global”:

```
mensaje_global = "Hola, ¿cómo estás?"

def saludar():
    print(mensaje_global)

saludar()
print(mensaje_global)   # Esto funciona bien
```

¡Vamos a la práctica! 

Ejercicios prácticos

Optativos | No entregables

Gestión de descuentos

Imaginá que en tu tienda querés implementar un sistema de descuentos automáticos. Vas a desarrollar un programa que permita calcular el precio final de un producto después de aplicar un descuento. Para hacerlo:

1. Crea una función que reciba como parámetros el precio original del producto y el porcentaje de descuento, y que retorne el precio final con el descuento aplicado.
2. Luego, pedí que se ingrese el precio y el porcentaje de descuento. Mostrá el precio final después de aplicar el descuento.

Ejercicios prácticos

Optativos | No entregables

Cálculo de promedio de ventas

Desarrollá un programa que permita calcular el promedio de ventas de la tienda. Para esto:

1. Creá una función que reciba como parámetro una lista de ventas diarias y devuelva el promedio de esas ventas.
2. Solicitá a la persona que ingrese las ventas de cada día durante una semana (7 días). Usá la función para calcular y mostrar el promedio de ventas al finalizar.