Unidad 4

Diccionarios

Los diccionarios en Python son una estructura de datos que permite almacenar su contenido en forma de clave y valor. A diferencia de las listas y tuplas donde accedíamos a un elemento mediante un índice, esta vez lo haremos por una clave única.

Veamos un ejemplo de un diccionario que almacena información de una persona:

```
persona = {"nombre": "Ana", "apellido": "Perez", "edad": 30}
```

Las características de un diccionario son:

- Las **claves** deben ser únicas e inmutables (como strings o tuplas).
- Los **valores** pueden ser de cualquier tipo.
- Es **mutable**: los valores pueden modificarse después de crearlos.

Operaciones básicas en diccionarios

Crear

```
dic = {"a": 5, "b": 10}
```

Obtener un valor mediante su clave

```
dic = {"a": 5, "b": 10}
print(dic["a"])  # Muestra 5
print(dic["c"])  # Da error porque la clave "c" no existe
```

Obtener un valor utilizando función get()

```
dic = {"a": 5, "b": 10}
print(dic.get("a"))  # Muestra 5
print(dic.get("c"))  # No da error, Muestra None
```

Modificar/Agregar

```
dic = {"a": 5, "b": 10}
dic["a"] = 20 # El valor 5 fue reemplazado por 20
dic["c"] = 15 # Como no existe la clave "c", se crea y se asigna el valor 15
print(dic) # Muestra el diccionario {'a': 20, 'b': 10, 'c': 15}
```

Eliminar

```
dic = {"a": 5, "b": 10}
valor_eliminado = dic.pop("a") # pop() elimina el valor y también lo devuelve
print(valor_eliminado) # Muestro el valor eliminado 5
print(dic) # Muestra el diccionario {'b': 10}
```

Funciones útiles

La variable de tipo diccionario tiene incorporada las siguientes funciones:

dic.keys() Devuelve una vista de las claves

dic.values() Devuelve los valores

dic.items() Devuelve pares clave-valor en una lista de tuplas

dic.update() Actualiza con otro diccionario

dic.clear() Elimina todo el contenido

dic.copy() Copia superficial del diccionario

dic.get(k, d) Devuelve valor o d si no existe k

Como iterar un diccionario

Al igual que con las tuplas y listas también podremos iterar un diccionario para acceder a sus valores, esto podremos realizarlo de dos maneras:

Mediante las claves del diccionario

```
dic = {"a": 1, "b": 2}
for clave in dic: # recorre solo las claves
    print(clave, dic[clave]) # utiliza la clave para acceder al valor
```

Mediante la función ítems()

```
dic = {"a": 1, "b": 2}
for clave, valor in dic.items(): # “Desarma” la lista de tuplas en clave valor
    print(clave, valor) # Muestro el valor sin necesidad de usar la clave
```

Funciones

Una función en Python es un bloque de código reutilizable que realiza una tarea específica. Se define una vez y puede llamarse múltiples veces.

Hasta el momento veníamos utilizando funciones incorporadas de Python como len() o funciones de conversión como int() y str(), veamos cómo crear nuestras propias funciones.

Ejemplo:

```
def saludar(nombre): # Defino la función saludar() que recibe como parámetro un nombre
    print("Hola", nombre) #Muestra por pantalla Hola Pedro

saludar("Pedro") # llamo a la función saludar con el parámetro "Pedro"
```

En el ejemplo creamos una función llamada saludar() que tiene como objetivo saludar al nombre que recibe como variable de entrada. El bloque de código dentro de la función toma la variable de entrada y la muestra por pantalla utilizando la función print().

Teniendo en cuenta lo anterior podemos describir los componentes de una función de la siguiente manera:

- **Nombre de la función:** es el nombre que utilizaremos para invocar la función. Ej: saludar, sumar, calcular_promedio, etc.
- **Parámetros (o argumentos):** Son los valores de entrada que recibe la función para luego utilizarlos en el bloque de código. Los parámetros pueden ser opcionales.
- **Bloque de código**: son las líneas de código que ejecutara la función invocada.
- **Valor de retorno**: Es el valor que retorna una función, también puede ser opcional.

En el caso anterior nuestra función no retornaba un valor, veamos un ejemplo donde si se retorne un valor.

Ejemplo:

```
def sumar(a,b): # Defino la función suma que recibe como parámetros dos variables
    return a + b # Retorno la suma de a + b

resultado = sumar(10,20) # Guardo el valor que retorna la función sumar() en resultado
print(resultado) #Muestro por pantalla el resultado
```

En este ejemplo creamos una función que tiene como objetivo sumar las variables de entrada y retornar el resultado de esa operación matemática. Invocamos la función sumar con los parámetros 10 y 20, el resultado de esa suma la almacenamos en una variable y luego lo mostramos por pantalla.

Argumentos de entrada

Los argumentos o variables de entrada que recibe una función pueden pasarse de varias formas, veamos a continuación cada una de ellas.

Argumentos por posición

Los argumentos posicionales son la forma más básica de pasar parámetros. La función sumar() que definimos anteriormente utiliza argumentos posicionales al momento de invocarla.

```
def sumar(a,b): # Defino la función suma que recibe como parámetros dos variables
    return a + b # Retorno la suma de a + b
```

Los parámetros a y b son argumentos posicionales por lo tanto debemos respetar el orden de los mismos al invocar la función.

Veamos otro ejemplo, para ello vamos a definir la funcion resta(), que retornara la resta de dos variables de entrada.

```
def resta(a,b): # Defino la función resta que recibe como parámetros dos variables
    return a - b # Retorno la resta de a - b

resultado = resta(20,10) # Guardo el valor que retorna la función resta() en resultado
print(resultado) #Muestro por pantalla el resultado
```

El valor que retorna la funcion resta() en este caso es 10 (20 – 10) pero si invertimos la posición de los argumentos el resultado será distinto. Por ejemplo, si invoco a la función resta de la siguiente manera:

```
resultado = resta(10,20) # Guardo el valor que retorna la función resta() en resultado
print(resultado) #Muestro por pantalla el resultado
```

Esto mostrara por pantalla -10 ya que la operación que se ejecuto fue 10 – 20.

Otro aspecto importante a tener en cuenta es la cantidad de argumentos que puede recibir una funcion, por ejemplo, la función suma() y resta() definidas anteriormente solo pueden recibir dos variables de entrada. Si invoco la función con menos o mas parámetros de los indicados entonces Python nos indicara un error posicional en la invocación.

```
resultado = resta(10,20,30) # error de posición, se esperaban solo dos argumentos
resultado = resta(10) # error de posición, se esperaba un argumento mas
```

Argumentos por nombre

Los argumentos por nombre utilizan el signo = para indicar a que variable debe asignarse el valor que recibe la función.

```
resta(a=20, b=10) # dará como resultado 10
resta(b=10, a=20) # también dará como resultado 10
```

Ambas formas de llamar a la función retornan el mismo resultado ya que utilizamos los nombres de las variables como argumentos. Esto tiene una gran ventaja respecto a los argumentos posicionales porque no es necesario respetar el orden siempre y cuando utilicemos los nombres correctos.

Sim embargo, hay que tener en cuenta que tampoco podremos pasar menos o mas argumentos de los que hayamos definido y tampoco podremos utilizar nombres de variables que no existan.

```
resta(a=20, b=10, c=30) # error de posición, se esperaban solo dos argumentos
resta(a=20) # error de posición, se esperaba un argumento mas
resta(a=20, c=10) # error de posicion porque "c" no esta definido
```

Argumentos por defecto

Los argumentos por defecto permiten definir un parámetro como opcional para que sea utilizado o no dependiendo del objetivo de la función.

Veamos como definir un argumento por defecto en la funciona suma().

```
def sumar(a, b, c=10): # La variable c tiene por defecto asignado el valor 10
    return a + b + c # Retorno la suma de a + b + c

sumar(10,20) # el resultado es 40 (10 + 20 + 10) utilizamos el valor por defecto de c
sumar(10,20,20) # el resultado es 50 (10 + 20 + 20) reemplazamos el valor por defecto de c
```

Esto es muy útil ya que podremos evitar los errores de posición si asignamos un argumento por defecto a cada variable de entrada.

Ejemplo:

```
def sumar(a=10, b=20, c=10):
    return a + b + c # Retorno la suma de a + b + c

sumar(10) # No da error de posición
sumar(10,20) # No da error de posición
sumar(10,20,20)  # No da error de posición
```