Temas

- Diccionarios
- Funciones

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.

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

- **Mediante la clave “nombre” accedo al valor “Ana”**
- **Mediante la clave “apellido” accedo al valor “Pérez”**
- **Mediante la clave “edad” accedo al valor 30**

```
print(persona["nombre"]) # Muestra por pantalla Ana
```

Diccionarios - Características

- 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.

Diccionarios – Operaciones Básicas

Podremos **crear** un diccionario de la siguiente manera:

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

Y **obtener** algunos de sus valores mediante sus claves.

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

Diccionarios – Operaciones Básicas

También podremos **obtener** un valor utilizando una clave y la **función get()**

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

Utilizar la **función get()** es útil cuando no sabemos si la clave existe en el diccionario, de esta forma **evitamos un error en la ejecución del programa.**

Diccionarios – Operaciones Básicas

Es posible **agregar o modificar** un valor dentro de un diccionario mediante su clave.

```
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}
```

Por último, también podremos **eliminar** un valor mediante su clave.

```
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}
```

Diccionarios – 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

Diccionarios – 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(), ahora vamos a 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"
```

Funciones

Los componentes de una función son los siguientes:

- **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.

Funciones – Retornar un valor

Veamos un ejemplo de una función donde se **retorne un valor** luego de invocarla.

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
```

Funciones – Argumentos de entrada

Los argumentos o variables de entrada que recibe una función pueden definirse de varias formas.

- **Argumentos por posición**
- **Argumentos por nombre**
- **Argumentos por defecto**

Funciones – Argumentos por posición

Los argumentos posicionales son la forma más básica de pasar parámetros. Veamos un ejemplo definiendo la función resta(), que al invocarla retornara la resta entre las 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 resultado de invocar esta función con los argumentos da 10 ya que la operación realizada fue 20 – 10.

Funciones – Argumentos por posición

¿Pero que sucedería si invertimos el orden de los argumentos al invocar la función?

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

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

Por lo tanto, debemos tener precaución al determinar la posición de los argumentos al invocar la función, de lo contrario obtendríamos un resultado que tal vez no sea el esperado.

Funciones – Argumentos por posición

Otro aspecto importante a tener en cuenta es la cantidad de argumentos que puede recibir una función, por ejemplo, la función suma() y resta() definidas anteriormente solo pueden recibir dos variables de entrada.

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

Si invoco la función con menos o mas parámetros de los necesarios entonces Python nos indicara un error posicional en la invocación.

Funciones – 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.

Funciones – Argumentos por nombre

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 posición porque "c" no esta definido
```

Funciones – 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.

```
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) no utilizamos el valor por defecto
```

Funciones – Argumentos por defecto

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

```
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
```