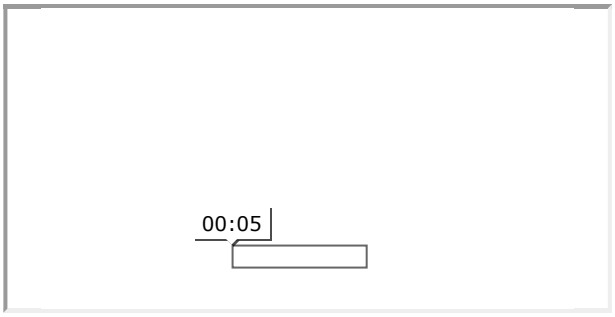
Módulo 3. Listas y diccionarios

Introducción

En módulos anteriores, se trabajó con variables de distintos tipos. Principalmente, se trabajó con números enteros, flotantes y cadenas de texto, lo cual parecería ser suficiente para procesar cualquier tipo de caso de uso, pero ¿qué sucedería si quisieras trabajar al mismo tiempo con 100 valores distintos? Una solución podría ser declarar 100 nuevas variables, aunque desde el punto de vista práctico esto no es viable, y la administración de las variables se vuelve relativamente compleja. Por otro lado, ¿cómo podrías agrupar distintos datos entre sí? Esto podría ser útil para relacionar información que pertenece a una misma entidad. Por ejemplo, imaginen una variable que indique que Carlos tiene 34 años y el pelo de color negro. Actualmente, resulta difícil lograr esto simplemente con los tipos de variables que se vieron hasta el momento. Por este motivo, se necesitan unas estructuras de datos un poco más complejas.

Video de inmersión



Unidad 1: Listas y tuplas

Tema 1. Definición de listas

Hasta el momento, se explicaron las variables que permiten almacenar un único valor a la vez. En el caso de que se necesite realizar un programa que requiera procesar decenas o cientos de datos al mismo tiempo, resulta poco práctico definir una variable por cada uno de estos datos. Para este caso de uso, existe un tipo de variable que se denomina **lista**.

Una lista permitirá almacenar múltiples valores en una misma variable, ya sean del mismo tipo o incluso de diferentes tipos: ya sean números enteros, flotantes, cadenas de texto, o incluso otra lista.

La manera de definir una lista es relativamente sencilla: se comienza con un corchete abierto, se definen los elementos separados por una coma, y se cierra con un corchete cerrado.

A continuación, se presenta un ejemplo de una lista de números:

Figura 1. Lista de números

```
nums = [1,5,13,65,3]
```

Fuente: elaboración propia.

Cabe destacar que los elementos de una lista no necesariamente deben ser únicos, sino que se pueden incluso repetir:

Figura 2. Repetir

```
random_var = ["Juan", 3, 52, "Pedro"]
```

Fuente: elaboración propia.

Al momento de imprimir en pantalla una lista, se puede hacer referencia a ella a través de la función print():

Figura 3. Función

```
print(nums)
```

Fuente: elaboración propia.

El resultado que se observa en pantalla es el siguiente:

Figura 4. Resultado

```
[1, 5, 13, 65, 3]
```

Fuente: elaboración propia.

A pesar de ello, en la mayoría de los casos no se buscará mostrar la lista completa, sino que se intentará acceder a un elemento en específico. Para poder realizar esto, se hace uso de los **índices**. Python definió como convención que el primer elemento de una lista lleva el índice 0, y los elementos siguientes tendrán el número de índice incrementado de a una unidad. Por lo tanto, en una lista de 7 elementos, el primer elemento tendrá el índice N.º 0, y el último elemento tendrá el índice N.º 6.

Conociendo este detalle sobre los índices, es hora de mostrar cómo acceder a un elemento en específico de la lista. La manera de hacerlo es referenciando al índice deseado entre corchetes.

Por ejemplo, haciendo referencia nuevamente a la lista de números, se desea ahora imprimir en pantalla el tercer elemento (el N.º 13). Teniendo en cuenta lo que se mencionó

previamente, dado que al primer elemento le corresponde el índice 0, al tercer elemento le corresponde entonces el índice 2:

Figura 5. Ejemplo

```
nums = [1, 5, 13, 65, 3]

print(nums[2])
```

Fuente: elaboración propia.

Si se ejecuta el *script* anterior, en pantalla se mostrará simplemente el número “13”.

Cabe aclarar que los índices negativos también son válidos; ellos se usan para comenzar a mostrar desde el último elemento hacia el inicio, en lugar del camino inverso:

Figura 6. Índices negativos

```
nums = [2,4,6,9,3]

print(nums[-1])
```

Fuente: elaboración propia.

El resultado que se muestra en pantalla, luego de ejecutar el *script* anterior, es simplemente el número “3”.

1. Pregunta de repaso

Los índices en una lista son útiles para hacer referencia a la posición de un elemento en la lista.

Verdadero

Falso

Justificación

Tema 2. Accediendo a elementos de una lista

Conociendo cómo definir una variable del tipo lista que contiene diversos elementos, y sabiendo cómo acceder a cada uno de estos elementos mediante el uso de índices, es momento de mostrar las distintas operaciones que pueden realizarse con las listas.

- **Suma de listas**

Al sumar dos listas a través del operador “suma” (+), el resultado que se obtiene es la concatenación de ambas listas. Esto quiere decir que una lista aparece al final de otra en una nueva variable (en caso se defina de esta manera):

Figura 7. Sumar

```
nums_1 = [1, 5, 13]

nums_2 = [10, 12,9]

res= nums_1 + nums_2

print(res)
```

Fuente: elaboración propia.

A continuación, se presenta el resultado que se mostró en pantalla al ejecutar el *script* anterior:

Figura 8. Resultado

```
[1, 5, 13, 10, 12, 9]
```

Fuente: elaboración propia.

- **Multiplicación de listas**

Al igual que en el ejemplo anterior, es posible utilizar el operador aritmético de multiplicación (*), y el resultado será la concatenación de la misma lista en una nueva variable, la cantidad de veces que se haya multiplicado:

Figura 9. Multiplicación

```
nums_1 = [1, 5, 13]

res= nums_1 * 2

print(res)
```

Fuente: elaboración propia.

Resultado mostrado en pantalla al ejecutar el *script* anterior:

Figura 10. Resultado

```
[1, 5, 13, 1, 5, 13]
```

Fuente: elaboración propia.

- **Reemplazar un valor en la lista**

Luego de haber definido una lista, es posible reemplazar un valor dentro de ella, haciendo referencia al índice del elemento, y asignando un nuevo valor.
En el siguiente ejemplo, se buscará reemplazar el cuarto elemento (por lo tanto, se hará referencia al índice 3) por el número 3:

Figura 11. Ejemplo

Fuente: elaboración propia.
Luego de ejecutar el *script*, si bien la lista original que había sido definida era [2, 4, 6, 9, 3], el resultado mostrado en pantalla será [2, 4, 6, 3, 3], ya que el cuarto elemento de la lista fue modificado.

- **Copiar valores de una lista**

Siguiendo el criterio del caso de uso anterior en el que se modifica un elemento existente de la lista, también es válido copiar valores entre distintos elementos de una lista. Por ejemplo, tomar el valor del cuarto elemento, y copiarlo en el segundo elemento. La manera en la que esto se realiza es la siguiente:

Figura 12. Copiar valores

Fuente: elaboración propia.
Al ejecutar el *script* anterior, si bien la lista original que había sido definida era [2, 4, 6, 9, 3], el resultado que se muestre en pantalla será [2, 9, 6, 9, 3], dado que el cuarto elemento se copió en el segundo.

Realizar las operaciones que se mencionaron anteriormente es extremadamente útil para actualizar información en una lista (debido a un cambio en el contexto), ya sea proveniente de una nueva variable que se especifica en el código, o utilizando los elementos ya definidos en las listas (utilizando operaciones de suma, multiplicación entre listas).

2. Pregunta de repaso

Al sumar dos listas utilizando el operador aritmético “+”, el resultado será una lista del mismo tamaño que la lista de mayor tamaño, y cada elemento será la suma de cada elemento de las listas que se encuentran en la misma ubicación.

Verdadero

Falso

Justificación

Tema 3. Operaciones con listas

Hasta este momento, las operaciones que se realizaban sobre las listas eran sobre listas definidas de manera estática en el código, cuya cantidad de elementos no variaba, y todas las modificaciones se realizaban sobre elementos existentes.
Existe la posibilidad de que las listas puedan expandir o reducir su tamaño durante el período de ejecución del *script*, de manera que se puedan ir agregando o eliminando valores a medida que sea necesario. Esto se hace a través del uso de funciones y métodos.

Si bien se estuvo trabajando hasta el momento con funciones que realizaban distintos objetivos, los métodos son un concepto nuevo.
Un método es un tipo de función que pertenece, específicamente, al tipo de dato con el que se trabaja. Las funciones, por otro lado, se pueden utilizar con variables para obtener

o mostrar un resultado, pero no pertenecen a un tipo de dato en particular, sino que al código en general. Los métodos cumplen también con el objetivo de mostrar y obtener resultados de una o varias variables, pero también permiten modificar el estado de la variable.

Adicionalmente, un método difiere en la forma de invocarse en comparación con una función. En general, una función se invoca de la siguiente manera:

resultado = función(argumento).

Mientras que un método se invoca de la siguiente manera:

resultado = data.metodo(argumento).

Esta breve introducción sobre funciones y métodos se hizo para comentar que estas son necesarias para poder trabajar con listas variables en tamaño. Se explicará el funcionamiento de las principales:

len() – Función.
append() – Método.
insert() – Método.
pop() – Método.

- **Función len()**

No siempre se conoce la cantidad de elementos que contiene una lista, ya sea porque su tamaño varía durante la ejecución del *script*, o porque la lista se creó en el programa (por ejemplo, leyendo datos de una fuente externa, como, por ejemplo, una planilla de Excel). A través de la función len(), se podrá conocer la cantidad de elementos que contiene una lista.

La manera en la que funciona es pasando la lista de interés como parámetro de esta función. A continuación, se dará un ejemplo de esto, y se imprimirá el resultado en pantalla a través de la función print():

Figura 13. Función print ()

```
nums_1 = [2,4,6,9,3]

print(len(nums_1))
```

Fuente: elaboración propia.

Luego de ejecutar el *script* anterior, se verá en la terminal el número “5”, que es el resultado de la función. Efectivamente, se puede observar que la lista “nums_1” contiene 5 elementos.

- **Método append()**

El método append() es extremadamente útil a la hora de añadir elementos al final de una lista. Su uso es muy simple: se hace referencia a la variable del tipo lista a la que se le desee añadir un elemento, seguido de un punto y la palabra append(). Entre paréntesis, se le pasará como argumento el elemento que se desee añadir al final de la lista:

Figura 14. Append ()

```
nums = [1,2,3,4]

nums.append(6)

print(nums)
```

Fuente: elaboración propia.

Al ejecutar el *script* anterior, se observa en la terminal el resultado siguiente:

Figura 15. Resultado

```
[1, 2, 3, 4, 6]
```

Fuente: elaboración propia.

Como pueden notar, al imprimir en pantalla la lista, se observa que se añadió el número 6 al final a través del método append().

A continuación, se incorporará una línea adicional al *script* anterior, añadiendo un elemento del tipo cadena de texto a la misma lista:

Figura 16. Resultado

```
nums = [1,2,3,4]

nums.append(6)
nums.append("fin")

print(nums)
```

Fuente: elaboración propia.

En este último ejemplo, además de haber añadido el número 6 al final de la lista que se declara al inicio, se realiza un paso adicional para añadir un elemento más. En este caso, se trata de una cadena de texto con el valor “fin”.

Al ejecutar este *script*, se observa en pantalla el siguiente resultado:

Figura 17. Resultado

Fuente: elaboración propia.

Contar con este método permite, además de los casos de uso que se presentaron previamente, definir una variable del tipo lista vacía al inicio del *script*, sin necesidad de cargarle elementos, y luego agregarle los elementos durante el período de ejecución del programa. A continuación, se dará un ejemplo de lo recién mencionado:

Figura 18. Resultado

Fuente: elaboración propia.

Como se observa, la lista nums no cuenta con ningún elemento al inicio del *script*. Posteriormente, se agrega el valor “1”, y, a continuación, el valor “3”. Por último, se imprime la lista en pantalla. Al ejecutar el programa, se observa el siguiente resultado en la terminal:

Figura 19. Resultado

Fuente: elaboración propia.

- **Método insert()**

De la misma manera que append() permite expandir el tamaño de una lista añadiendo elementos hacia su final, el método insert() permite añadir elementos en cualquier ubicación dentro de una lista. Para ello, requiere que se especifiquen como parámetros, por un lado, el índice en el cual se desea insertar el valor, y, por el otro lado, el elemento que se desea insertar.

A continuación, se demuestra, en un *script* de ejemplo, cómo se inserta un elemento del tipo cadena de texto en la tercera posición (por lo tanto, índice 2) de una lista de números:

Figura 20. Lista de números

Fuente: elaboración propia.

Al ejecutar el *script* anterior, se observa en pantalla el siguiente resultado:

Figura 21. Resultado

Fuente: elaboración propia.

- **Método pop()**

Python permite modificar el tamaño de una lista no solo añadiendo elementos, sino que también eliminando. Para poder eliminar un elemento de la lista, contamos con el método pop(). Su uso es relativamente sencillo: consta simplemente de pasar como parámetro el índice del elemento que se desea eliminar de la lista.

En el siguiente ejemplo, se definirá una lista de cuatro elementos, y, a través del método pop(), se eliminará el primer elemento de la lista (índice 0):

Figura 22. Resultado

Fuente: elaboración propia.

Al ejecutar el *script* anterior, se observa en pantalla el siguiente resultado:

Figura 23. Resultado

Fuente: elaboración propia.

En lugar de tener una lista de cuatro elementos como se definió al inicio del *script*, se observa que se muestra una lista de tres elementos, y que falta el primer elemento original, ya que se eliminó.

3. Pregunta de repaso

¿Cuáles de estas opciones corresponden a métodos válidos de listas?

insert() -

remove().

pop() -

append() -

start().

Justificación

Tema 4. Definición y diferencia con tuplas

En Python, una tupla es una estructura de datos muy similar a una lista. De hecho, la diferencia puede ser prácticamente imperceptible: en lugar de definir la variable entre corchetes, una tupla se define entre paréntesis; pero esta no es la única diferencia, sino que cuenta con una característica fundamental: una tupla es inmutable. Esto quiere decir que, una vez que se definen los elementos que la componen, no se pueden modificar.

Al leer esto, puede parecer que no es conveniente usar las tuplas… ¿Por qué usaría una estructura de datos que pierde funcionalidades en comparación con una lista? En realidad, no se están perdiendo funcionalidades, sino que dichas características son útiles en diferentes escenarios: por ejemplo, por la manera en la que se almacenan en memoria, las tuplas son más rápidas de procesar que las listas, lo cual reduce el tiempo de ejecución del código, y lo hace, de esta manera, más eficiente para los casos de uso en los que no se desea modificar el contenido de este tipo de estructura de datos. Por otro lado, el hecho de que sea inmutable permite contar con un seguro antiescritura.

A continuación, se observa un ejemplo de definición de una tupla, y luego se imprime en pantalla:

Figura 24. Definición de tupla

```
tupla_nums = (2,5,8,1)

print(tupla_nums)
```

Fuente: elaboración propia.

Al ejecutar el *script* anterior, se observa el siguiente resultado en pantalla:

Figura 25. Resultado

Fuente: elaboración propia.

La forma de referirse a un elemento dentro de una tupla, es exactamente igual a como se realiza en una lista. Puede resultar confuso, pero es importante destacar que, si bien una tupla se define entre paréntesis, al momento de referirse a uno de sus elementos a través del índice, se hace entre corchetes al igual que en una lista.

En el siguiente ejemplo, se imprimirá en la terminal el segundo elemento de la tupla (por lo tanto, el índice 1):

Figura 26. Segundo elemento

Fuente: elaboración propia.

Al ejecutar el *script* anterior, en la pantalla se muestra tal cual lo esperado el número 5.

4. Preguntas de repaso

Dado que las tuplas cuentan con la característica de no poder modificarse, siempre será más conveniente trabajar con listas.

Verdadero

Falso

Justificación

Unidad 2: Diccionarios

Tema 1. Definición de diccionarios

En algunas ocasiones, se necesita agrupar valores. Por ejemplo, si se quisiera mantener un registro de personas, y para cada una de ellas se quisieran guardar datos como el nombre, la edad y el género, con el conocimiento adquirido hasta el momento, esto podría resolverse con una lista para cada uno de los datos, si se tuviera una lista para nombres, una para edades y otra para géneros. Luego, debería ser necesario asociar que el primer elemento de cada lista corresponde a una persona, el segundo a otra persona, y así sucesivamente. Si bien esto es una solución válida a la necesidad, resulta muy poco práctica.

Para poder resolver la situación anterior de una manera más óptima, existe una estructura de datos denominada **diccionarios**.

Los diccionarios, a diferencia de las listas, no agrupan valores aislados, sino que almacenan, en cada elemento, pares de llaves y valores. Un ejemplo de llave podría ser “edad”, y el valor “24”, haciendo referencia a 24 años. Cada uno de los elementos del diccionario (separados por coma, al igual que en las listas y tuplas) almacenará uno de estos pares, y todos sus elementos corresponderán a una misma entidad. Para el caso del nombre y la edad, se podría definir un diccionario por persona de la que se quiera mantener un registro, para poder agrupar todos estos datos en una misma variable por cada persona.
La manera de definir un diccionario en Python es entre llaves (“{}”). En cada uno de sus elementos, la llave estará definida con un nombre entre comillas, seguido de dos puntos (“:”), y el valor asociado a esa llave. El valor puede tratarse de cualquier tipo de variable: número entero, flotante, cadena de texto, booleano, etc.

A continuación, se demuestra lo que se comentó anteriormente, y se define un diccionario con el nombre de variable “persona_1”, que contiene distintos datos sobre una persona:

Figura 27. Persona_1

```
persona_1 = {"nombre": "Juan", "apellido": "Lopez", "edad": 42}
```

Fuente: elaboración propia.

5. Pregunta de repaso

Se pretende que el alumno observe la siguiente imagen en la que se está definiendo un diccionario, y que indique por qué el código arroja error al ejecutar el *script*:

Figura 28. Diccionario

Fuente: elaboración propia.

Se deben reemplazar las comas por punto y comas.

Se deben reemplazar los corchetes por llaves

Se deben iniciar todos los nombres de las llaves con mayúscula.

Está faltando la línea de código para imprimir el diccionario en pantalla.

Tema 2. Accediendo a elementos de un diccionario

Si se recuerda que un elemento en un diccionario se compone por una llave y un valor, es posible referenciar a la llave del elemento, para obtener el valor que se le asignó. De la misma manera en la que en una lista o tupla se obtenía el valor de un elemento haciendo referencia al índice en el que se localizaba, en un diccionario no se trabaja con el índice, sino que se trabaja con la llave.

Tomando como referencia el diccionario que se definió anteriormente, se buscará imprimir en pantalla la edad de la persona, haciendo referencia a la llave “edad” para obtener el valor del elemento:

Figura 29. Edad

Fuente: elaboración propia.

Como se puede observar, si bien el diccionario se define entre llaves, para hacer referencia a la llave del elemento, se hace entre corchetes.

Al ejecutar el *script* anterior, se observa el siguiente resultado en pantalla:

Figura 30. Edad

Fuente: elaboración propia.

Como se esperaba, se muestra el valor asignado a la llave “edad” en el diccionario “persona_1”.

6. Preguntas de repaso

Para poder hacer referencia a un elemento en particular del diccionario, se debe especificar el índice, es decir, la ubicación del elemento dentro del diccionario.

Verdadero

Falso

Justificación

Tema 3. Operaciones con diccionarios

Luego de definir un diccionario con sus respectivos elementos, o declarar la variable como un diccionario vacío (diccionario ejemplo = {}), hay tres operaciones fundamentales para realizar con ellos: modificar un elemento existente, agregar un nuevo elemento, o eliminar un elemento. Estas operaciones se utilizan de manera frecuente siempre que se desee actualizar información del diccionario. Esto puede darse, por ejemplo, cuando hay que agregar un nuevo dato (podría ser el caso de una nueva persona que se registra a un servicio), eliminar un dato (una persona se da de baja de una suscripción, y el diccionario representa los suscriptores), o modificar un dato existente (alguien desea corregir su fecha de nacimiento).

- **Modificar un elemento existente**

Para modificar un elemento existente en un diccionario, se debe hacer referencia a la llave del elemento, y, a través del operador “=”, asignarle un nuevo valor. En el siguiente ejemplo, se tomará el diccionario con el que se viene trabajando, y se modificará el nombre de la persona:

Figura 31. “=”

Fuente: elaboración propia.

Se observa que, si bien al momento de definir el diccionario el valor asignado a la llave “nombre” era “Juan”, posteriormente se le asigna el valor “Pedro” siguiendo los pasos que se indican anteriormente.

Al ejecutar el *script* anterior, se observa en pantalla el siguiente resultado, lo cual confirma el cambio de valor de la llave “nombre”:

Figura 32. “nombre”

Fuente: elaboración propia.

- Agregar un nuevo elemento

Para agregar un nuevo elemento, el proceso es exactamente el mismo que cuando se modifica un valor existente. La única diferencia es que la llave a la que se hace referencia no está presente aún en el diccionario, sino que es el elemento que se desea agregar.

A continuación, se dará un ejemplo agregando la llave “género” con valor “masculino” al diccionario con el que se viene trabajando:

Figura 33. “género”

Fuente: elaboración propia.

Al ejecutar el *script* anterior, se observa el siguiente resultado en pantalla:

Figura 34. Resultado

Fuente: elaboración propia.

Si bien el diccionario no se definió originalmente incluyendo la llave género, se observa que la llave se agrega siguiendo los pasos comentados, y, al imprimir el diccionario, se observa en terminal que la llave se adicionó al final del diccionario.

- **Eliminar un elemento**

Es posible también eliminar un elemento de un diccionario. Esto se realiza a través de la palabra clave “del”, seguida del diccionario haciendo referencia a la llave del elemento que se desea eliminar.

Continuando con el ejemplo anterior, se demostrará, a continuación, cómo eliminar el elemento que contiene la llave “nombre”:

Figura 35. eliminar “nombre”

Fuente: elaboración propia.

Al ejecutar el *script* anterior, se puede observar en pantalla:

Figura 36. Resultado

Fuente: elaboración propia.

Por lo tanto, podemos concluir con que el elemento se eliminó tal cual se esperaba.

7. Pregunta de repaso

Considerando que se desea modificar el valor de un elemento de un diccionario (definido en una variable denominada “dict_personas”), cuya llave es “nombre”,

¿a través de qué línea de código podría realizarse?

dict_personas[“nombre"] = "Pedro"

dict_personas(“nombre”) = “Pedro”.

dict_personas[1] = “Pedro”.

dict_personas(1) = “Pedro”.

Justificación

Tema 4. Métodos de diccionarios

Al igual que con las listas, los diccionarios cuentan con métodos que permiten interactuar con sus elementos para realizar distintas acciones sobre estos datos, ya sea para mostrarlos o para modificarlos.

- **Método keys()**

El método keys() permite acceder a un listado de las llaves de un diccionario. Esto será útil, por ejemplo, para verificar la existencia de una llave dentro de él:

Figura 37. keys()

Fuente: elaboración propia.

Al ejecutar el *script* anterior, se puede observar el siguiente resultado en la terminal:

Figura 38. Resultado

Fuente: elaboración propia.

- **Método values()**

El método values() permite acceder a un listado de los valores de un diccionario. Esto será útil, por ejemplo, para verificar la existencia de un valor en particular dentro del diccionario, o para realizar un procesamiento de datos de acuerdo con los valores obtenidos:

Figura 39. values ()

```
persona_1 = {"nombre": "Juan", "apellido": "Lopez", "edad": 42}

print(persona_1.values())
```

Fuente: elaboración propia.

Al ejecutar el *script* anterior, se puede observar el siguiente resultado en la terminal:

Figura 40. Resultado

```
dict_values(['Juan', 'Lopez', 42])
```

Fuente: elaboración propia.

- **Método items()**

El método items() devuelve una lista de tuplas; cada una de ellas contiene, en el primer elemento, la llave del ítem, y, en el segundo elemento, el valor asignado:

Figura 41. items ()

```
persona_1 = {"nombre": "Juan", "apellido": "Lopez", "edad": 42}

print(persona_1.items())
```

Fuente: elaboración propia.

Al ejecutar el *script* anterior, se puede observar el siguiente resultado en la terminal:

Figura 42. Resultado

```
dict_items([('nombre', 'Juan'), ('apellido', 'Lopez'), ('edad', 42)])
```

Fuente: elaboración propia.

- **Método pop ()**

El método pop() cumple exactamente la misma función que con las listas: eliminar un elemento en la posición determinada. De esta manera, en el caso de los diccionarios, se

proporciona la llave del elemento que se va a eliminar:

Figura 43. pop()

```
persona_1 = {"nombre": "Juan", "apellido": "Lopez", "edad": 42}

persona_1.pop("nombre")

print(persona_1)
```

Fuente: elaboración propia.

Al ejecutar el *script* anterior, se puede observar el siguiente resultado en la terminal:

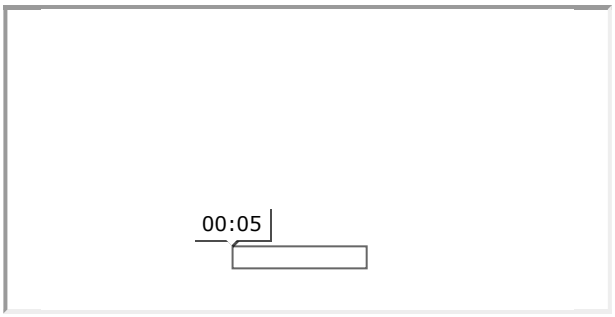
Figura 44. Resultado

```
{'apellido': 'Lopez', 'edad': 42}
```

Fuente: elaboración propia.

Como se puede notar, se eliminó el elemento que poseía la llave “nombre”.

Video de habilidades



Preguntas de habilidades

1. Para hacer referencia al primer elemento de una lista se debe utilizar el índice “1”.

Verdadero

Falso

Justificación

2. Para poder hacer referencia a un elemento de un diccionario se debe especificar su índice entre llaves.

Verdadero

Falso

Justificación

3. ¿Cuál de los siguientes caracteres sirve para definir una lista en Python?

Llaves

Paréntesis

Barras

Corchetes

Justificación

4. Es posible agregar elementos a un diccionario luego de haberlo definido.

Verdadero

Falso

Justificación

5. ¿Qué método de listas permite agregar elementos al final de una lista?

Append()

Pop()

Add()

Insert()

Justificación

Cierre

En esta lectura, se cubrieron dos de las estructuras de datos más importantes del lenguaje de programación Python: listas y diccionarios. Dominar y tener práctica sobre estos conceptos es imprescindible, dado que, muy probablemente, este tipo de variables estén

presentes en la gran mayoría de nuestros *scripts*. Conociendo ahora cómo trabajar y realizar operaciones básicas con este tipo de estructuras, es posible avanzar a la siguiente lectura para conocer cómo aplicar distintos tipos de lógicas para evaluar condiciones que definamos sobre estas mismas estructuras de datos, y poder recorrerlas de distintas maneras para leer la información disponible.

Glosario