# Tecnicatura Universitaria en Programación
# Programación II

## UNIDAD N° 1: Lenguaje Python - Sintaxis y Elementos básicos y avanzados

## Índice

# Colecciones

Una colección permite agrupar varios objetos bajo un mismo nombre. Por ejemplo, si necesitamos almacenar en nuestro programa los nombres de los alumnos de un curso de programación, será más conveniente ubicarlos a todos dentro de una misma colección de nombre alumnos, en lugar de crear los objetos alumno1, alumno2, etc.

# Listas

Una lista es una colección de elementos en un orden particular. Puedes hacer una lista que incluya las letras del alfabeto, los dígitos del 0 al 9 o los nombres de todas las personas de tu familia. Puede poner lo que quiera en una lista, y los elementos de su lista no tienen que estar relacionados de ninguna manera en particular.

Porque una lista generalmente contiene más de un elemento, es una buena idea hacer que el nombre de su lista sea plural, como letters, digits o names.

En Python, los corchetes [ ] indican una lista y los elementos individuales de la lista están separados por comas.


Si le pide a Python que imprima una lista, Python devuelve su representación de la lista, incluidos los corchetes: Debido a que este no es el resultado que desea que vean sus usuarios, debemos acceder a los elementos individuales en una lista.

## Acceso a elementos en una lista

Las listas son colecciones ordenadas, por lo que puede acceder a cualquier elemento de una lista diciéndole a Python la posición o el índice del elemento deseado. Para acceder a un elemento de una lista, escriba el nombre de la lista seguido del índice del elemento entre corchetes.

```
example.py > ...
1   bicycles = ['trek', 'cannondale', 'redline', 'specialized']
2   print(bicycles[0])

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
PS C:\PII\Prueba> python example.py
trek
```

Este es el resultado que desea que vean sus usuarios: una salida limpia y con un formato ordenado.

***Nota:*** *Las posiciones de índice comienzan en 0, no en 1. Python considera que el primer elemento de una lista está en la posición 0, no en la posición 1.*

Python tiene una sintaxis especial para acceder al último elemento de una lista. Si solicita el elemento en el índice -1, Python siempre devuelve el último elemento de la lista. Esta sintaxis es bastante útil, porque a menudo querrá acceder a los últimos elementos de una lista sin saber exactamente cuántos elementos tiene dicha lista.

Esta convención se extiende también a otros valores de índice negativos. El índice -2 devuelve el segundo elemento desde el final de la lista, el índice -3 devuelve el tercer elemento desde el final, y así sucesivamente.

**Uso de valores individuales de una lista**

Puede usar valores individuales de una lista como lo haría con cualquier otra variable. Por ejemplo, puede usar f-strings para crear un mensaje basado en un valor de una lista.

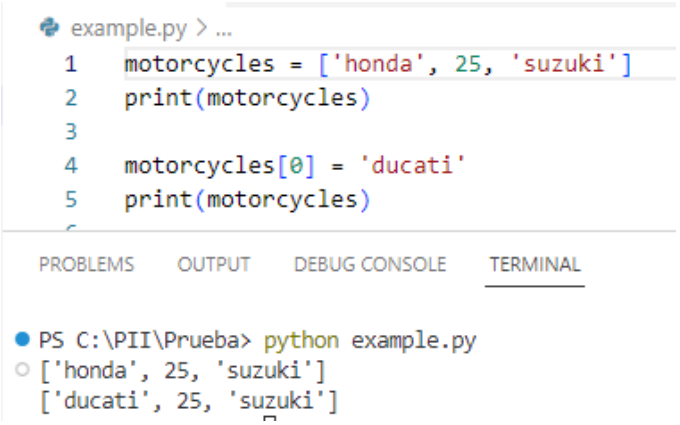
```
example.py > ...
1   bicycles = ['trek', 'cannondale', 'redline', 'specialized']
2   message = f"My first bicycle was a {bicycles[0].title()}."
3
4   print(message)

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
PS C:\PII\Prueba> python example.py
My first bicycle was a Trek.
```

## Modificación, adición y eliminación de elementos

La mayoría de las listas que cree serán dinámicas , lo que significa que creará una lista y luego agregará y eliminará elementos de ella a medida que su programa sigue su curso.
Por ejemplo, puede crear un juego en el que un jugador tenga que disparar a los extraterrestres desde el cielo. Puede almacenar el conjunto inicial de alienígenas en una lista y luego eliminar un alienígena de la lista cada vez que uno es derribado. Cada vez que aparece un nuevo extraterrestre en la pantalla, lo agregas a la lista. Tu lista de alienígenas aumentará y disminuirá en longitud a lo largo del juego.

## Modificación de elementos en una lista

La sintaxis para modificar un elemento es similar a la sintaxis para acceder a un elemento en una lista. Para cambiar un elemento, use el nombre de la lista seguido del índice del elemento que desea cambiar y luego proporcione el nuevo valor que desea que tenga ese elemento.

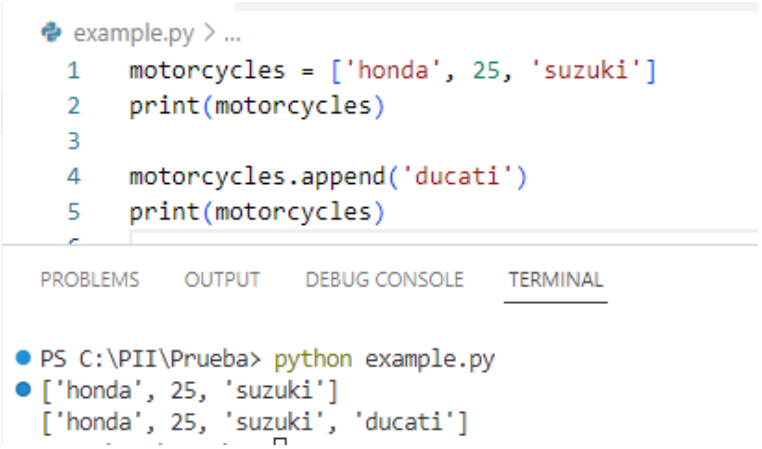



## Adición de elementos a una lista

Es posible que desee agregar un nuevo elemento a una lista por muchas razones. Por ejemplo, es posible que desee hacer que aparezcan nuevos extraterrestres en un juego, agregar nuevos datos a una visualización o agregar nuevos usuarios registrados a un sitio web que ha creado.

Python proporciona varias formas de agregar nuevos datos a las listas existentes.

La forma más sencilla de agregar un nuevo elemento a una lista es agregar el elemento al final de la misma.




El método ***append()*** facilita la creación dinámica de listas. Por ejemplo, puede comenzar con una lista vacía y luego agregar elementos a la lista mediante una serie de llamadas a ***append()***.

```
example.py > ...
1   motorcycles = []
2
3   motorcycles.append('honda')
4   motorcycles.append('yamaha')
5   motorcycles.append('suzuki')
6
7   print(motorcycles)
```

La creación de listas de esta manera es muy común, porque a menudo no sabrá los datos que sus usuarios desean almacenar en un programa hasta después de que el programa se esté ejecutando. Para que sus usuarios tengan el control, comience definiendo una lista vacía que contendrá los valores de los usuarios. Luego agregue cada nuevo valor proporcionado a la lista que acaba de crear.

**Insertar elementos en una lista**

Puede agregar un nuevo elemento en cualquier posición de su lista utilizando el método insert(). Para ello, especifique el índice del nuevo elemento y el valor del nuevo elemento:

```
example.py > ...
1   motorcycles = ['honda', 'yamaha', 'suzuki']
2
3   motorcycles.insert(0, 'ducati')
4   print(motorcycles)
5
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

PS C:\PII\Prueba> python example.py
['ducati', 'honda', 'yamaha', 'suzuki']
```

Esta operación desplaza todos los demás valores de la lista una posición a la derecha.

**Eliminación de elementos de una lista mediante del**

A menudo, desea eliminar un elemento o un conjunto de elementos de una lista. Por ejemplo, cuando un jugador dispara a un alienígena desde el cielo, lo más probable es que quieras eliminarlo de la lista de alienígenas activos. O cuando un usuario decide cancelar su cuenta en una aplicación web que usted creó, querrá eliminar a ese usuario de la lista de usuarios activos. Puede eliminar un elemento según su posición en la lista o según su valor.

Si conoce la posición del elemento que desea eliminar de una lista, puede usar la declaración ***del***. Puede eliminar un elemento de cualquier posición en una lista si conoce su índice.

```
example.py > ...
1   motorcycles = ['honda', 'yamaha', 'suzuki']
2   print(motorcycles)
3
4   del motorcycles[1]
5   print(motorcycles)

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\PII\Prueba> python example.py
['honda', 'yamaha', 'suzuki']
['honda', 'suzuki']
```

Ya no se puede acceder al valor que se eliminó de la lista después de usar la declaración ***del***.

**Eliminación de un elemento mediante el método pop()**

A veces querrá usar el valor de un elemento después de eliminarlo de una lista. Por ejemplo, es posible que desee obtener la posición x e y de un alienígena que acaba de ser derribado, para poder dibujar una explosión en esa posición. En una aplicación web, es posible que desee eliminar un usuario de una lista de miembros activos y luego agregar ese usuario a una lista de miembros inactivos.

El método ***pop()*** elimina el último elemento de una lista, pero le permite trabajar con ese elemento después de eliminarlo. El término pop proviene de pensar en una lista como una pila de elementos y sacar un elemento de la parte superior de la pila. En esta analogía, la parte superior de una pila corresponde al final de una lista.

```
example.py > ...
1   motorcycles = ['honda', 'yamaha', 'suzuki']
2   print(motorcycles)
3
4   popped_motorcycle = motorcycles.pop()
5   print(motorcycles)
6   print(popped_motorcycle)

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\PII\Prueba> python example.py
['honda', 'yamaha', 'suzuki']
['honda', 'yamaha']
suzuki
```

**Reventar elementos desde cualquier posición en una lista**

Puede usar ***pop()*** para eliminar un elemento de cualquier posición en una lista incluyendo el índice del elemento que desea eliminar entre paréntesis.

Recuerde que cada vez que usa pop(), el elemento con el que trabaja ya no se almacena en la lista.

```
example.py > ...
1   motorcycles = ['honda', 'yamaha', 'suzuki']
2
3   second_owned = motorcycles.pop(1)
4   print(f"The second motorcycle I owned was a {second_owned.title()}.")
5
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

PS C:\PII\Prueba> python example.py
The second motorcycle I owned was a Yamaha.
```

**pop() vs del**

Si no está seguro de si usar la declaración del o el método pop(), aquí tiene una forma simple de decidir: cuando desee eliminar un elemento de una lista y no usar ese elemento de ninguna manera, use del; si desea usar un elemento a medida que lo elimina, use el pop().

**Eliminación de un artículo por valor**

A veces no sabrá la posición del valor que desea eliminar de una lista. Si solo conoce el valor del artículo que desea eliminar, puede usar el remove()método.

```
example.py > ...
1   motorcycles = ['honda', 'yamaha', 'suzuki', 'ducati']
2   print(motorcycles)
3
4   motorcycles.remove('ducati')
5   print(motorcycles)
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

PS C:\PII\Prueba> python example.py
['honda', 'yamaha', 'suzuki', 'ducati']
['honda', 'yamaha', 'suzuki']
```

También puede usar el método remove() para trabajar con un valor que se está eliminando de una lista.

```
example.py > ...
1   motorcycles = ['honda', 'yamaha', 'ducati', 'suzuki', 'ducati']
2   print(motorcycles)
3
4   too_expensive = 'ducati'
5   motorcycles.remove(too_expensive)
6   print(motorcycles)
7   print(f"\nA {too_expensive.title()} is too expensive for me.")
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

PS C:\PII\Prueba> python example.py
['honda', 'yamaha', 'ducati', 'suzuki', 'ducati']
['honda', 'yamaha', 'suzuki', 'ducati']

A Ducati is too expensive for me.
```

El método remove() elimina solo la primera aparición del valor que especifique. Si existe la posibilidad de que el valor aparezca más de una vez en la lista, deberá usar un bucle para asegurarse de que se eliminen todas las apariciones del valor. (Luego se verá cómo).

## Organizar una lista

A menudo, sus listas se crearán en un orden impredecible, porque no siempre puede controlar el orden en que los usuarios proporcionan sus datos. Aunque esto es inevitable en la mayoría de las circunstancias, con frecuencia querrá presentar su información en un orden particular. A veces querrá conservar el orden original de su lista, y otras veces querrá cambiar el orden original.
Python proporciona varias formas diferentes de organizar sus listas, según la situación.

**Ordenar una lista de forma permanente con el método sort()**

El método de Python ***sort()*** hace que sea relativamente fácil ordenar una lista. Imagina que tenemos una lista de autos y queremos cambiar el orden de la lista para almacenarlos alfabéticamente.

```
example.py > ...
1   cars = ['bmw', 'audi', 'toyota', 'subaru']
2   cars.sort()
3   print(cars)
4

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\PII\Prueba> python example.py
['audi', 'bmw', 'subaru', 'toyota']
```

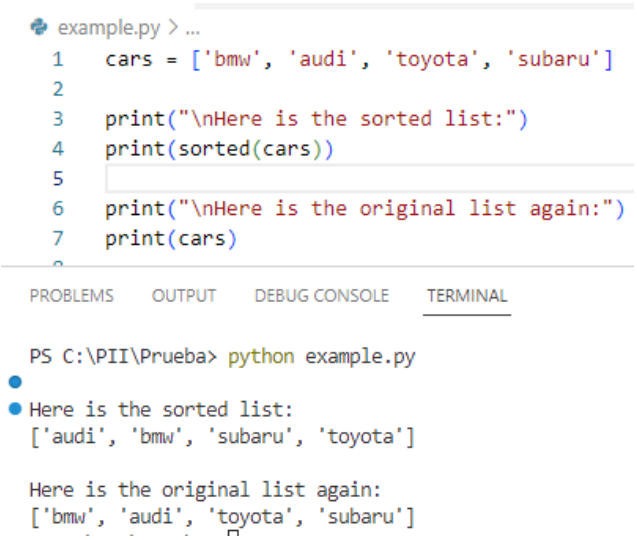
El método ***sort()*** cambia el orden de la lista de forma permanente. Los autos ahora están en orden alfabético y nunca podremos volver al orden original.

También puede ordenar esta lista en orden alfabético inverso pasando el argumento ***reverse=True*** al método ***sort().***

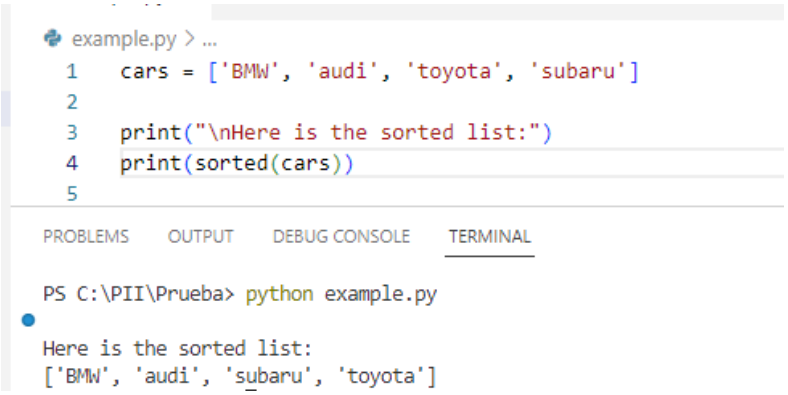
```
example.py > ...
1   cars = ['bmw', 'audi', 'toyota', 'subaru']
2   cars.sort(reverse=True)
3   print(cars)
4

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\PII\Prueba> python example.py
['toyota', 'subaru', 'bmw', 'audi']
```

## Ordenar una lista temporalmente con la función sorted()

Para mantener el orden original de una lista pero presentarla ordenada, puede utilizar la función ***sorted()***. La función ***sorted()*** le permite mostrar su lista en un orden particular, pero no afecta el orden real de la lista.




***Nota:*** *Ordenar una lista alfabéticamente es un poco más complicado cuando todos los valores no están en minúsculas. Hay varias formas de interpretar las letras mayúsculas al determinar un orden de clasificación, y especificar el orden exacto puede ser más complejo de lo que queremos tratar en este momento.*




## Imprimir una lista en orden inverso

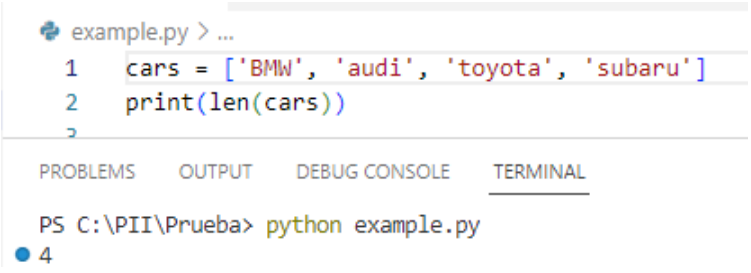
Para invertir el orden original de una lista, puede utilizar el método ***reverse()***. Si originalmente almacenamos una lista de autos en orden cronológico según modelo, podríamos reorganizar fácilmente la lista en orden cronológico inverso.

El método ***reverse()*** no ordena alfabéticamente hacia atrás; simplemente invierte el orden de la lista.

El método ***reverse()*** cambia el orden de una lista de forma permanente, pero puede volver al orden original en cualquier momento aplicando ***reverse()*** a la misma lista una segunda vez.

## Longitud de una lista

Puede encontrar rápidamente la longitud de una lista utilizando la función ***len()***.
Por ejemplo la función ***len()*** es útil cuando se necesita identificar la cantidad de alienígenas que aún deben ser derribados en un juego, determinar la cantidad de datos que debe administrar en una visualización o averiguar la cantidad de usuarios registrados en un sitio web. entre otras tareas.

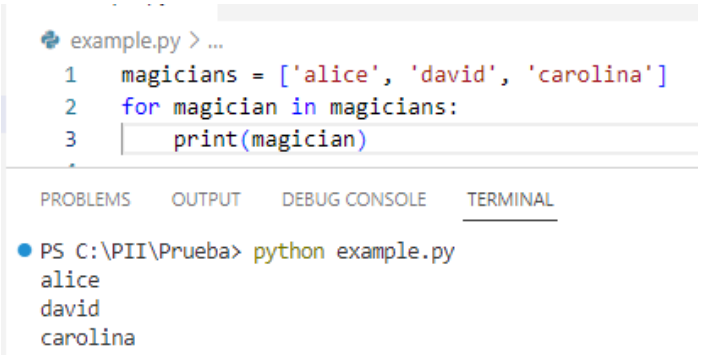



## Bucle a través de una lista completa

A menudo querrá recorrer todas las entradas de una lista, realizando la misma tarea con cada elemento.

Por ejemplo, en un juego, es posible que desee mover todos los elementos de la pantalla en la misma cantidad. En una lista de números, es posible que desee realizar la misma operación estadística en cada elemento. O tal vez desee mostrar cada título de una lista de artículos en un sitio web.

Cuando desee realizar la misma acción con todos los elementos de una lista, puede utilizar el ciclo for de Python.




***for magician i magicians*** → Esta línea le dice a Python que recupere el primer valor de la lista magicians y lo asocie con la variable magician. Y repite la misma operatoria hasta que no haya más elementos en la lista.

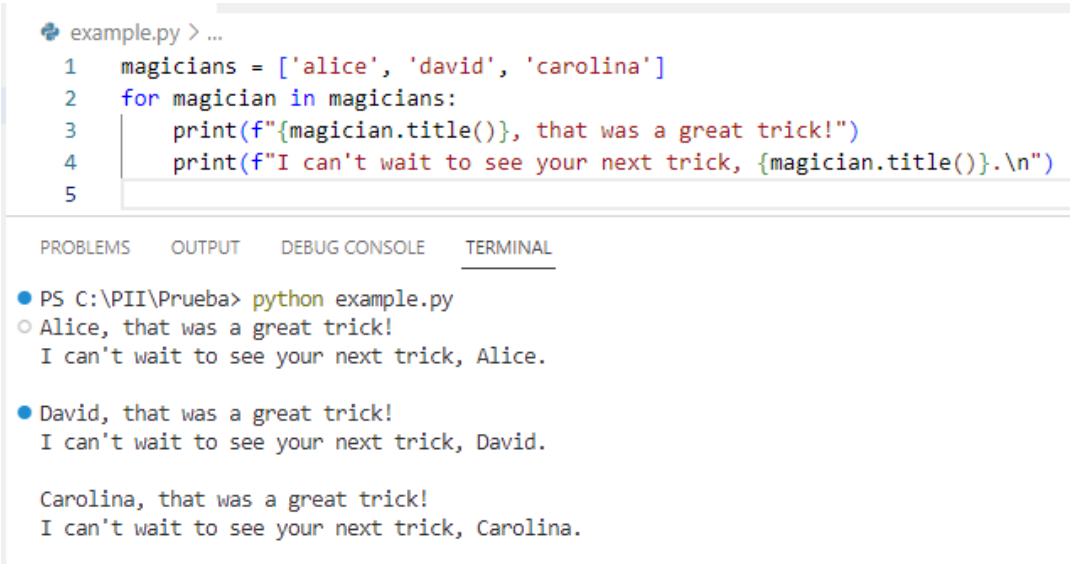
Cuando utilice bucles por primera vez, tenga en cuenta que el conjunto de pasos se repite una vez para cada elemento de la lista, sin importar cuántos elementos haya en la lista. Si tiene un

millón de elementos en su lista, Python repite estos pasos un millón de veces y, por lo general, muy rápido.

También tenga en cuenta que al escribir sus propios bucles for puede elegir cualquier nombre que desee para la variable temporal que se asociará con cada valor en la lista. Sin embargo, es útil elegir un nombre significativo que represente un solo elemento de la lista. Por ejemplo:
***for cat in cats:***
***for dog in dogs:***
***for item in list_of_items:***

Estas convenciones de nomenclatura pueden ayudarlo a seguir la acción que se realiza en cada elemento dentro de un bucle for. El uso de nombres singulares y plurales puede ayudarlo a identificar si una sección de código funciona con un solo elemento de la lista o con la lista completa.

Puede escribir tantas líneas de código como desee en el bucle for. Cada línea indentada que sigue a la línea for magician in magicians se considera dentro del bucle , y se ejecuta una vez para cada valor de la lista.




## Evitar errores de sangría

Python utiliza la sangría para determinar cómo se relaciona una línea o un grupo de líneas con el resto del programa. En los ejemplos anteriores, las líneas que imprimían mensajes a los magos individuales formaban parte del bucle for porque estaban indentadas. El uso de indentado de Python hace que el código sea muy fácil de leer. Básicamente, utiliza espacios en blanco para forzarte a escribir código con un formato limpio y una estructura visual clara.

En los programas de Python más largos, notará bloques de código sangrados en algunos niveles diferentes. Estos niveles de sangría lo ayudan a obtener una idea general de la organización general del programa.

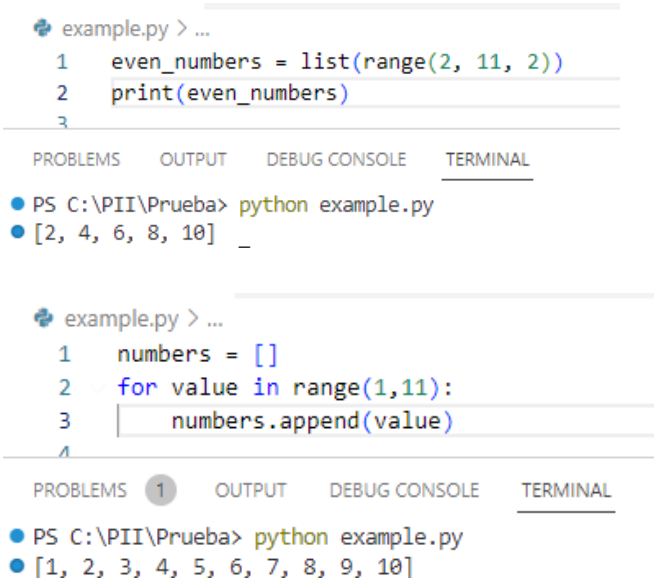
A medida que comience a escribir código que se base en una sangría adecuada, deberá estar atento a algunos errores de sangría comunes . Por ejemplo, las personas a veces sangran líneas de código que no necesitan sangría u olvidan sangrar líneas que necesitan sangría. Ver

ejemplos de estos errores ahora lo ayudará a evitarlos en el futuro y corregirlos cuando aparezcan en sus propios programas.

## Hacer listas numéricas

Existen muchas razones para almacenar un conjunto de números. Por ejemplo, deberá realizar un seguimiento de las posiciones de cada personaje en un juego y es posible que desee para realizar un seguimiento de las puntuaciones más altas de un jugador también. En las visualizaciones de datos, casi siempre trabajará con conjuntos de números, como temperaturas, distancias, tamaños de población o valores de latitud y longitud, entre otros tipos de conjuntos numéricos.

La función de Python ***range()*** facilita la generación de una serie de números. La misma se explicó anteriormente.




## Lista de comprensiones

El enfoque descrito anteriormente para generar la lista consistía en utilizar tres o cuatro líneas de código. Una lista de comprensión le permite generar esta misma lista en una sola línea de código.

Una lista por comprensión combina el bucle for y la creación de nuevos elementos en una sola línea, y agrega automáticamente cada elemento nuevo.

```
example.py > ...
1   numbers = [value for value in range(1, 11)]
2   print(numbers)
3
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

PS C:\PII\Prueba> python example.py
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

## Cortar una lista

Para hacer un corte, especifica el índice del primer y último elemento con el que desea trabajar. Al igual que con la función ***range()***, Python detiene un elemento antes del segundo índice que especifique.

```
example.py > ...
1   players = ['charles', 'martina', 'michael', 'florence', 'eli']
2   sub_players = players[0:3]
3   print(sub_players)
4
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

PS C:\PII\Prueba> python example.py
['charles', 'martina', 'michael']
```

Si omite el primer índice, Python inicia automáticamente su segmento al principio de la lista.

***players[:3]***

Una sintaxis similar funciona si desea un segmento que incluya el final de una lista. Por ejemplo, si desea todos los elementos desde el tercer elemento hasta el último elemento, puede comenzar con el índice 2 y omitir el segundo índice.

***players[2:]***

## Copiar una lista

A menudo, se desea comenzar con una lista existente y crear una lista completamente nueva basada en la primera.

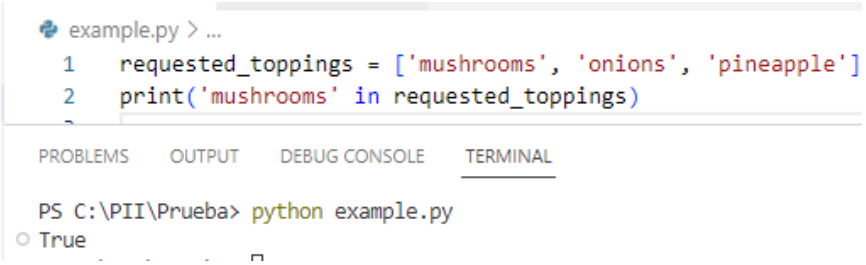
Para copiar una lista, puede crear un segmento que incluya la lista original completa omitiendo el primer índice y el segundo índice ( [:] ). Esto le dice a Python que haga una porción que comience en el primer elemento y termine en el último elemento, produciendo una copia de la lista completa.

```
example.py > ...
1   my_foods = ['pizza', 'falafel', 'carrot cake']
2   friend_foods = my_foods[:]
```

## Comprobar si un valor está en una lista

A veces es importante comprobar si una lista contiene un determinado valor antes de realizar una acción. Por ejemplo, es posible que desee verificar si ya existe un nuevo nombre de usuario en una lista de nombres de usuario actuales antes de completar el registro de alguien en un sitio web.
Para averiguar si un valor en particular ya está en una lista, use la palabra clave in.

Consideremos un código que podrías escribir para una pizzería. Haremos una lista de ingredientes que un cliente ha solicitado para una pizza y luego verificaremos si ciertos ingredientes están en la lista.




La palabra clave ***in*** le dice a Python que verifique la existencia de 'mushrooms' en la lista requested_toppings. Esta técnica es bastante poderosa porque puede crear una lista de valores esenciales y luego verificar fácilmente si el valor que está probando coincide con uno de los valores de la lista.

## Comprobar si un valor no está en una lista

Otras veces, es importante saber si un valor no aparece en una lista. Puede utilizar la palabra clave ***not in*** en esta situación.

Por ejemplo, considere una lista de usuarios que tienen prohibido comentar en un foro. Puede verificar si un usuario ha sido prohibido antes de permitir que esa persona envíe un comentario.

### Comprobar que una lista no está vacía

Verifiquemos si una lista de ingredientes solicitados para una pizza se encuentra vacía.

```
example.py > ...
1   requested_toppings = []
2
3   if requested_toppings:
4       for requested_topping in requested_toppings:
5           print(f"Adding {requested_topping}.")
6       print("\nFinished making your pizza!")
7   else:
8       print("Are you sure you want a plain pizza?")
9
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\PII\Prueba> python example.py
Are you sure you want a plain pizza?
```

## Tuplas

Las listas funcionan bien para almacenar colecciones de elementos que pueden cambiar a lo largo de la vida de un programa. Sin embargo, a veces querrá crear una lista de elementos que no se pueden cambiar.  Las tuplas te permiten hacer precisamente eso.

Python se refiere a valores que no pueden cambiar como inmutables , y una lista inmutable se llama tupla.

En comparación con las listas, las tuplas son estructuras de datos simples. Úselos cuando desee almacenar un conjunto de valores que no deben cambiarse a lo largo de la vida de un programa.

### Definición de una tupla

Una tupla se parece a una lista, excepto que usa paréntesis en lugar de corchetes. Una vez que define una tupla, puede acceder a elementos individuales utilizando el índice de cada elemento, tal como lo haría con una lista.

Por ejemplo, si tenemos un rectángulo que siempre debe tener un tamaño determinado, podemos asegurarnos de que su tamaño no cambie poniendo las dimensiones en una tupla.

```
example.py > ...
1   dimensions = (200, 50)
2   print(dimensions[0])
3   print(dimensions[1])
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\PII\Prueba> python example.py
200
50
```

Si intentamos cambiar el valor de la primera dimensión, Python devuelve un error de tipo. Debido a que estamos tratando de alterar una tupla, lo que no se puede hacer con ese tipo de objeto, Python nos dice que no podemos asignar un nuevo valor a un elemento en una tupla.

```
example.py > ...
1   dimensions = (200, 50)
2   dimensions[0] = 250
3
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\PII\Prueba> python example.py
Traceback (most recent call last):
  File "C:\PII\Prueba\example.py", line 2, in <module>
    dimensions[0] = 250
    ~~~~~~~~~~^^^
TypeError: 'tuple' object does not support item assignment
```

Las tuplas se definen técnicamente por la presencia de una coma; los paréntesis los hacen parecer más ordenados y legibles.

```
example.py > ...
1   dimensions = 200, 50
2   print(dimensions[0])
3   print(dimensions[1])
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\PII\Prueba> python example.py
200
50
```

Si desea definir una tupla con un elemento, debe incluir una coma final.

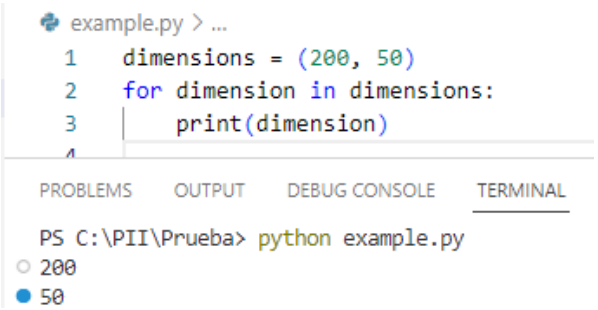
```
example.py > ...
1   dimensions = (200,)
2   print(dimensions[0])
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\PII\Prueba> python example.py
200
```

***A menudo no tiene sentido construir una tupla con un elemento.***

## Recorriendo todos los valores en una tupla

Puede recorrer todos los valores en una tupla usando se puede hacer tal como lo hizo con una lista con un bucle for.




### Escribir sobre una tupla

Aunque no puede modificar una tupla, puede asignar nuevos valores a la variable que representa la tupla, es decir volver a asignarla.

## Diccionarios

Comprender los diccionarios le permite modelar una variedad de objetos del mundo real con mayor precisión. Podrá crear un diccionario que represente a una persona y luego almacenar toda la información que desee sobre esa persona. Puede almacenar su nombre, edad, ubicación, profesión y cualquier otro aspecto de una persona que pueda describir.

Podrá almacenar dos tipos de información que se pueden combinar, como una lista de palabras y sus significados, una lista de nombres de personas y sus números favoritos, una lista de montañas y sus elevaciones, etc.

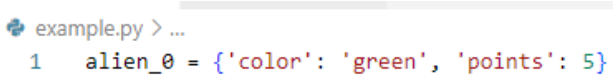
### Un diccionario simple

Un diccionario en Python es una colección de pares ***clave-valor***. Cada clave está conectada a un valor y puede usar una clave para acceder al valor asociado con esa clave. El valor de una clave puede ser un número, una cadena, una lista o incluso otro diccionario. De hecho, puede usar cualquier objeto que pueda crear en Python como un valor en un diccionario.

En Python, un diccionario se envuelve entre llaves ***{}*** con una serie de pares ***clave-valor*** dentro de las llaves.

| **Sintaxis diccionario** |
| --- |
| mi_diccionario = {"key1":<value1>,"key2":<value2>, … ,"keyN":<valueN>} |

**Cada par de clave-valor es denominado como elemento (item).**

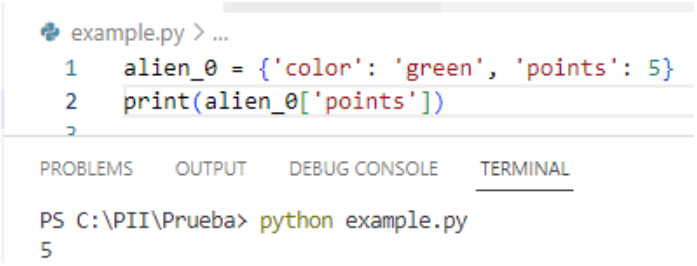



Un par clave-valor es un conjunto de valores asociados entre sí. Cuando proporciona una clave, Python devuelve el valor asociado con esa clave. Cada clave está conectada a su valor por dos

puntos, y los pares clave-valor individuales están separados por comas. Puede almacenar tantos pares clave-valor como desee en un diccionario.

## Acceso a valores en un diccionario

Para obtener el valor asociado con una clave, ingrese el nombre del diccionario y luego coloque la clave dentro de un conjunto de corchetes.

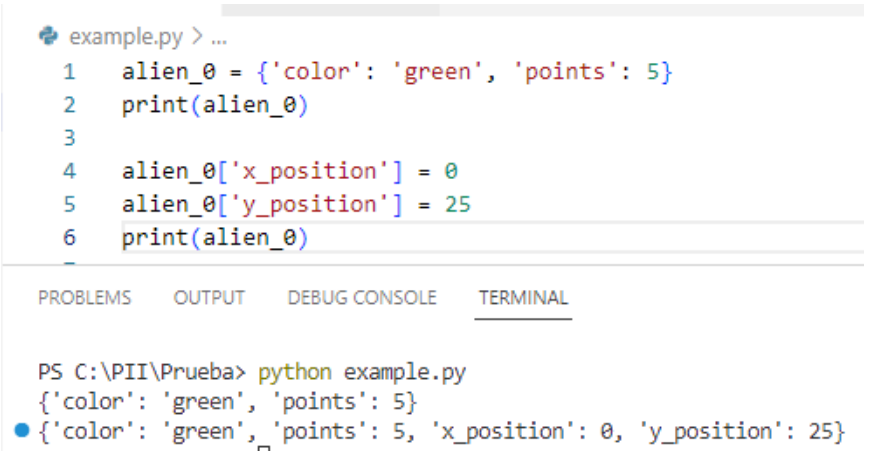



## Modificación, adición y eliminación de elementos

Los diccionarios son estructuras dinámicas , lo que significa que se agregarán y eliminarán elementos del mismo.

### Adición de nuevos pares clave-valor

Se puede agregar nuevos pares clave-valor a un diccionario en cualquier momento. Para agregar un nuevo par clave-valor, debe dar el nombre del diccionario seguido de la nueva clave entre corchetes, junto con el nuevo valor.




Los diccionarios conservan el orden en que fueron definidos. Cuando imprima un diccionario o recorra sus elementos, verá los elementos en el mismo orden en que se agregaron al diccionario.

### Comenzando con un diccionario vacío

A veces es conveniente, o incluso necesario, comenzar con un diccionario vacío y luego agregarle cada elemento nuevo. Para comenzar a llenar un diccionario vacío, defina un diccionario con un conjunto vacío de llaves y luego agregue cada par clave-valor.

Por lo general, usará diccionarios vacíos cuando almacene datos proporcionados por el usuario en un diccionario o cuando escriba código que genere una gran cantidad de pares clave-valor automáticamente.

### Modificación de valores en un diccionario

Para modificar un valor en un diccionario, proporcione el nombre del diccionario con la clave entre corchetes y luego el nuevo valor que desea asociar con esa clave.

```
example.py > ...
1   alien_0 = {'color': 'green'}
2   alien_0['color'] = 'yellow'
3   print(f"The alien is now {alien_0['color']}.")
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\PII\Prueba> python example.py
The alien is now yellow.
```

### Eliminación de pares clave-valor

Cuando ya no necesite una parte de la información almacenada en un diccionario, puede usar la declaración ***del*** para eliminar por completo un par clave-valor. Todo lo que necesita es el nombre del diccionario y la clave que desea eliminar. Tenga en cuenta que el par clave-valor eliminado se elimina de forma permanente.

```
example.py > ...
1   alien_0 = {'color': 'green', 'points': 5}
2   print(alien_0)
3   del alien_0['points']
4   print(alien_0)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\PII\Prueba> python example.py
{'color': 'green', 'points': 5}
{'color': 'green'}
```

## Usando get() para acceder a los valores

El uso de claves entre corchetes para recuperar el valor que le interesa de un diccionario puede causar un problema potencial: si la clave que solicita no existe, obtendrá un error.

El método ***get()*** requiere una clave como primer argumento. Como segundo argumento opcional, puede pasar el valor que se devolverá si la clave no existe.

```
example.py > ...
1   alien_0 = {'color': 'green', 'speed': 'slow'}
2
3   point_value = alien_0.get('points', 'No point value assigned.')
4   print(point_value)

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\PII\Prueba> python example.py
No point value assigned.
```

***Nota:*** *Si omite el segundo argumento en la llamada a get() y la clave no existe, Python devolverá el valor None.*

## Bucle a través de un diccionario

Un solo diccionario de Python puede contener solo unos pocos pares clave-valor o millones de pares. Debido a que un diccionario puede contener grandes cantidades de datos, Python le permite recorrer un diccionario.

Los diccionarios se pueden utilizar para almacenar información de diversas formas; por lo tanto, existen varias formas diferentes de recorrerlos. Puede recorrer todos los pares clave-valor de un diccionario, sus claves o sus valores.

### Bucle a través de todos los pares clave-valor

Para hacerlo, puede recorrer el diccionario usando un bucle for

```
example.py > ...
1   user = {
2       'username': 'efermi',
3       'first': 'enrico',
4       'last': 'fermi',
5       }
6   for key, value in user.items():
7       print(f"\nKey: {key}")
8       print(f"Value: {value}")

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\PII\Prueba> python example.py

Key: username
Value: efermi

Key: first
Value: enrico

Key: last
Value: fermi
```

En la primera mitad de la instrucción for ***[ for key, value ]***, las dos variables que contendrán la clave y el valor en cada par clave-valor, puede definirse con los nombres que desee.
En la segunda mitad se incluye el nombre del diccionario seguido del método ***items()***, que devuelve una secuencia de pares clave-valor.
Luego, el bucle for asigna cada uno de estos pares a las dos variables proporcionadas.

## Recorriendo todas las claves de un diccionario

El método ***keys()*** es útil cuando no se necesita trabajar con los valores en un diccionario, sólo necesitamos recuperar las claves.

Supongamos un diccionario que favorite_languages que contenga el lenguaje (valor) favorito para cada empleado de una empresa.

```
example.py > ...
1   favorite_languages = {
2       'jen': 'python',
3       'sarah': 'c',
4       'edward': 'rust',
5       'phil': 'python',
6       }
7
8   for name in favorite_languages.keys():
9       print(name.title())

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\PII\Prueba> python example.py
Jen
Sarah
Edward
Phil
```

Recorrer las claves de un diccionario en realidad es el comportamiento predeterminado cuando se recorre un diccionario, por lo que este código tendría exactamente el mismo comportamiento que el anterior.

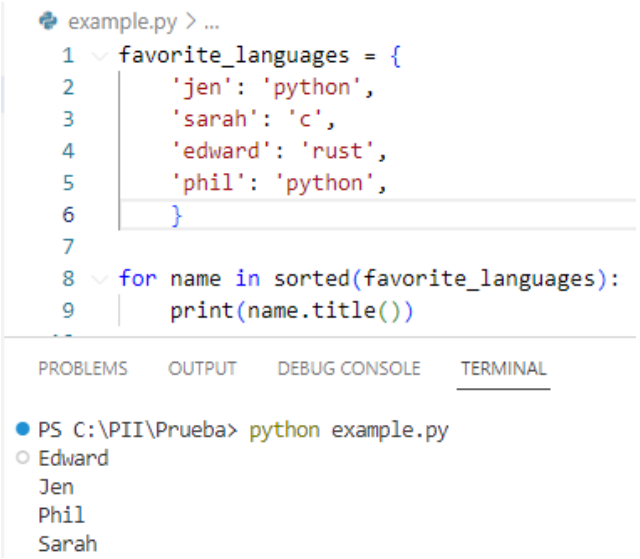
```
example.py > ...
1   favorite_languages = {
2       'jen': 'python',
3       'sarah': 'c',
4       'edward': 'rust',
5       'phil': 'python',
6       }
7
8   for name in favorite_languages:
9       print(name.title())

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\PII\Prueba> python example.py
Jen
Sarah
Edward
Phil
```

**Puede elegir usar el método keys() explícitamente para hacer que su código sea más fácil de leer, o puede omitirlo si lo desea.**

## Recorriendo las claves de un diccionario en un orden particular

Al recorrer un diccionario devuelve los elementos en el mismo orden en que se insertaron. A veces, sin embargo, querrá recorrer un diccionario en un orden diferente.

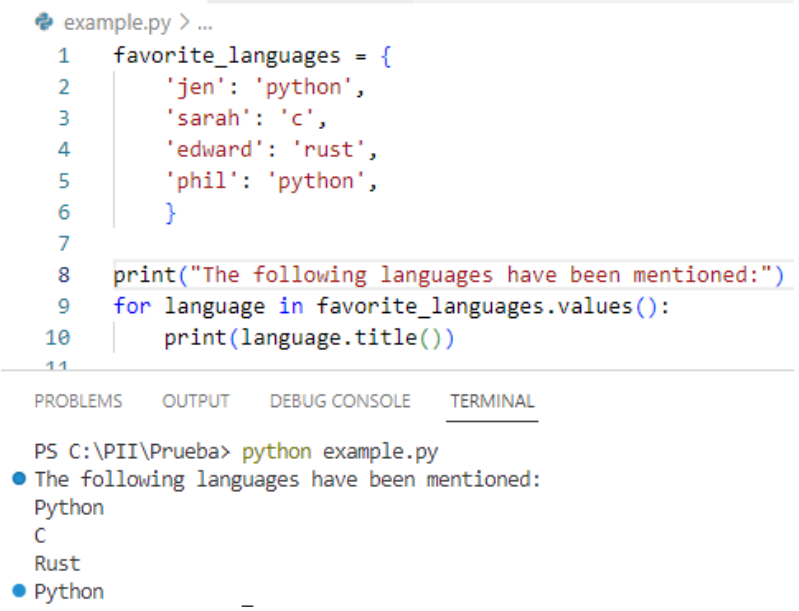
Una forma de hacer esto es ordenar las claves a medida que se devuelven en el forciclo. Puede usar la función ***sorted()*** para obtener una copia de las claves en orden.




## Recorriendo todos los valores en un diccionario

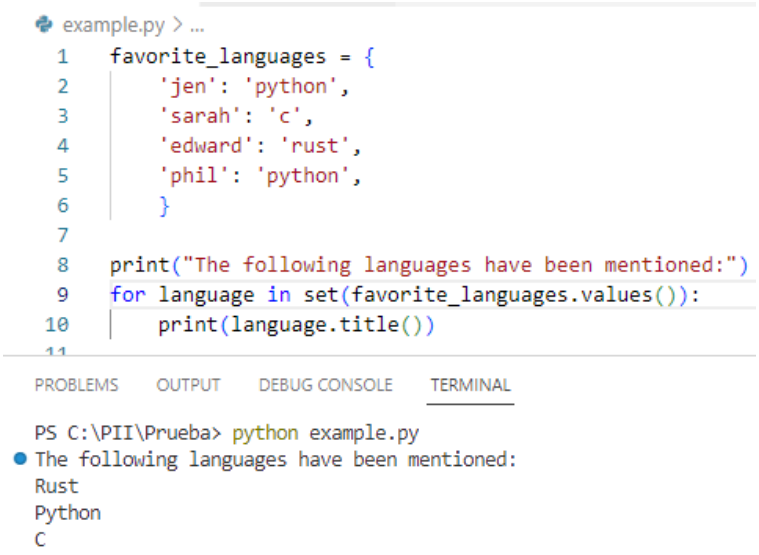
Si está interesado principalmente en los valores que contiene un diccionario, puede usar el método ***values()*** para devolver una secuencia de valores sin ninguna clave.

Por ejemplo, digamos que simplemente queremos una lista de todos los lenguajes elegidos en nuestra encuesta de lenguajes de programación, sin el nombre del empleado que lo eligió.



Este enfoque extrae todos los valores del diccionario sin verificar si hay repeticiones. Esto podría funcionar bien con una pequeña cantidad de valores, pero en una encuesta con una gran cantidad de encuestados, resultaría en una lista muy repetitiva.

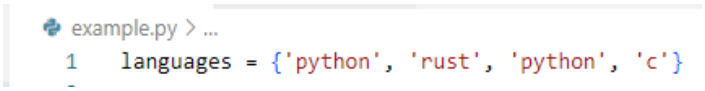
Para ver cada idioma lenguaje sin repetición, podemos usar un conjunto. Un conjunto es una colección en la que cada elemento debe ser único. Utilizamos la función ***set()***.




Cuando se envuelve con ***set()*** una colección de valores que contiene elementos duplicados, Python identifica los elementos únicos en la colección y crea un conjunto a partir de esos elementos.

## Conjuntos

Como mencionamos, un conjunto es una colección de valores no repetitivos. Puedes construir un conjunto directamente usando llaves y separando los elementos con comas.




Es fácil confundir conjuntos con diccionarios porque ambos están entre llaves.

Cuando ves llaves pero no pares clave-valor, probablemente estés viendo un conjunto. A diferencia de las listas y los diccionarios, los conjuntos no conservan elementos en ningún orden específico.

Al igual que otras colecciones, sus miembros pueden ser de diversos tipos, no obstante, un conjunto no puede incluir objetos mutables como listas, diccionarios, e incluso otros conjuntos.

Para generar un conjunto vacío, directamente creamos una instancia de la clase set, ya que por defecto, la asignación con { } crea un diccionario vacío.

```
3   conjunto = set()
4
5   diccionario = {}
```

## Modificación, adición y eliminación de elementos

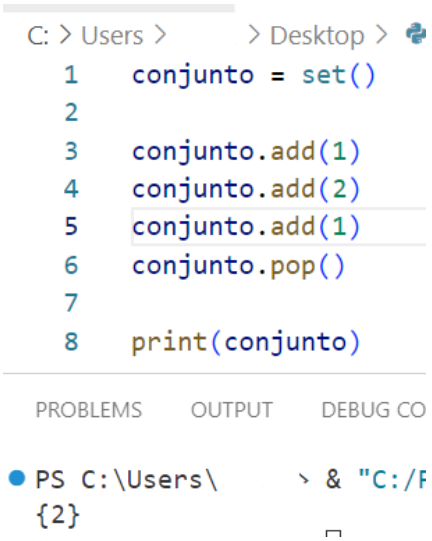
Los conjuntos son objetos mutables, lo que significa que se agregarán y eliminarán elementos.

Vía el método ***add()*** podemos añadir un elemento pasado como argumento al conjunto. Si el mismo ya existe en el conjunto, no es duplicado y simplemente se ignora.

Vía el método ***discard()*** podemos remover un elemento pasado como argumento. Si el elemento pasado como argumento al método ***discard()*** no está dentro del conjunto es simplemente ignorado.

Otra forma de remover un elemento del conjunto es mediante el método ***remove( )***, que opera de forma similar pero en caso de que el elemento pasado cómo argumento no se encuentre en el conjunto, lanza la excepción KeyError.

El método ***pop( )*** retorna un elemento en forma aleatoria (no podría ser de otra manera ya que los elementos no están ordenados). El mismo lanza una excepción KeyError si el conjunto está vacío.




## Operaciones con Conjuntos

Los set en Python tiene gran cantidad de métodos, por lo que recomendamos investigar cada uno de ellos, aquí un listado con alguno de ellos:

```
C: > Users >        > Desktop > app.py > ...
  1   conjunto1 = {1}
  2   conjunto2 = {1,2}
  3
  4   print(conjunto1.union(conjunto2))
  5   print(conjunto1.intersection(conjunto2))
  6   print(conjunto1.difference(conjunto2))
  7   print(conjunto1.symmetric_difference(conjunto2))
  8   print(conjunto1.isdisjoint(conjunto2))
  9   print(conjunto1.issubset(conjunto2))
 10   print(conjunto1.issuperset(conjunto2))
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

PS C:\Users\      > & "C:/Program Files/Python311/python
{1, 2}
{1}
set()
{2}
False
True
False
```

# Anidamiento

A veces querrá almacenar varios diccionarios en una lista o una lista de elementos como valor en un diccionario. Esto se llama anidamiento .
Puede anidar diccionarios dentro de una lista, una lista de elementos dentro de un diccionario o incluso un diccionario dentro de otro diccionario. El anidamiento es una característica poderosa.

## Una lista de diccionarios

Considere una lista de extraterrestres en la que cada extraterrestre sea un diccionario de información sobre ese extraterrestre. Por ejemplo, el siguiente código crea una lista de tres extraterrestres.

```
example.py > ...
  1   alien_0 = {'color': 'green', 'points': 5}
  2   alien_1 = {'color': 'yellow', 'points': 10}
  3   alien_2 = {'color': 'red', 'points': 15}
  4   aliens = [alien_0, alien_1, alien_2]
  5
  6   for alien in aliens:
  7       print(alien)
```

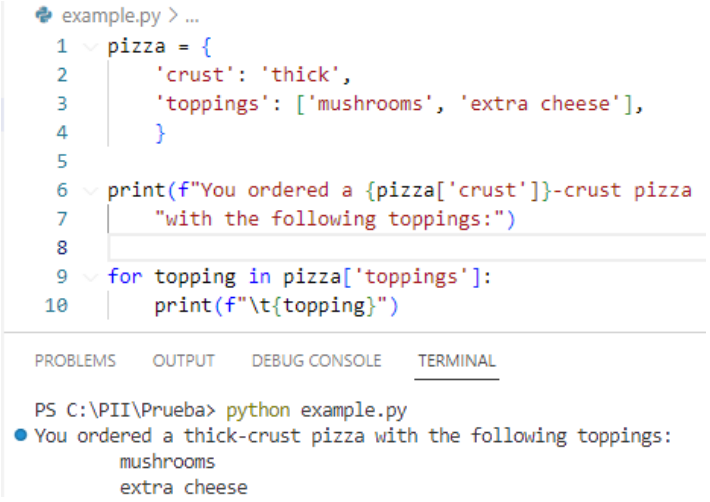
```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

PS C:\PII\Prueba> python example.py
{'color': 'green', 'points': 5}
{'color': 'yellow', 'points': 10}
{'color': 'red', 'points': 15}
```

Primero creamos tres diccionarios, cada uno representando un extraterrestre diferente. Almacenamos cada uno de estos diccionarios en una lista llamada aliens . Finalmente, recorremos la lista e imprimimos cada extraterrestre.

Es común almacenar varios diccionarios en una lista cuando cada diccionario contiene muchos tipos de información sobre un objeto.

## Una lista en un diccionario

En lugar de poner un diccionario dentro de una lista, a veces es útil poner una lista dentro de un diccionario.
Por ejemplo, considere cómo podría describir una pizza que alguien está ordenando. Si tuviera que usar solo una lista, todo lo que realmente podría almacenar es una lista de los ingredientes de la pizza. Con un diccionario, una lista de ingredientes puede ser solo un aspecto de la pizza que estás describiendo.




**Puede anidar una lista dentro de un diccionario en cualquier momento que desee asociar más de un valor con una sola clave en un diccionario.**

***Nota:*** *No debe anidar listas y diccionarios demasiado profundo. Si está anidando elementos mucho más profundo que 1 o 2 niveles, lo más probable es que haya una forma más sencilla de resolver el problema.*

## Un diccionario en un diccionario

Puede anidar un diccionario dentro de otro diccionario, pero su código puede complicarse rápidamente cuando lo hace.

Por ejemplo, si tiene varios usuarios para un sitio web, cada uno con un nombre de usuario único, puede usar los nombres de usuario como claves en un diccionario. A continuación, puede almacenar información sobre cada usuario mediante el uso de un diccionario como el valor asociado con su nombre de usuario.

```python
users = {
    'aeinstein': {
        'first': 'albert',
        'last': 'einstein',
        'location': 'princeton',
        },

    'mcurie': {
        'first': 'marie',
        'last': 'curie',
        'location': 'paris',
        },
    }
```

Observe que la estructura del diccionario de cada usuario es idéntica. Aunque Python no lo requiere, esta estructura facilita el trabajo con los diccionarios anidados. Si el diccionario de cada usuario tuviera claves diferentes, el código dentro de por ejemplo el ciclo for sería complicado.

## Usar un bucle while con listas y diccionarios

Un bucle for es eficaz para recorrer una lista, pero no debe modificar una lista dentro de un bucle for porque Python tendrá problemas para realizar un seguimiento de los elementos de la lista.

Para modificar una lista mientras trabaja en ella, utilice un bucle while. El uso de bucles while con listas y diccionarios le permite recopilar, almacenar y organizar muchas entradas para examinarlas e informar sobre ellas más adelante.

```python
pets = ['dog', 'cat', 'dog', 'goldfish', 'cat', 'rabbit', 'cat']
print(pets)

while 'cat' in pets:
    pets.remove('cat')

print(pets)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\PII\Prueba> python example.py
['dog', 'cat', 'dog', 'goldfish', 'cat', 'rabbit', 'cat']
['dog', 'dog', 'goldfish', 'rabbit']
```

## Desempaquetado de Tuplas y listas

En algunas ocasiones, nos podemos encontrar con una tupla, o lista, que contiene varios valores de los cuales solamente nos interesan unos pocos. Por lo que extraer solamente estos valores y quedarnos con los necesarios puede simplificar los posteriores trabajos. Esto es algo

que se puede conseguir mediante el **desempaquetado** en Python de una tupla o lista. Una tarea más sencilla de lo que parece.

El desempaquetado de todos los valores de una tupla o lista se realiza como sigue

```
program.py > ...
1   lista = ["primer", 25, [1, 2, 3]]
2   a, b, c = lista
```

A la hora de desempaquetar en Python una tupla o lista hay que tener en cuenta que es necesario indicar tantas variables como elementos de la lista. En caso contrario se producirá un error.

```
program.py > ...
1   lista = ["primer", 25, [1, 2, 3]]
2   a, b = lista

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\PII\py\python-practice1> python program.py
Traceback (most recent call last):
  File "C:\PII\py\python-practice1\program.py", line 2, in <module>
    a, b = lista
    ^^^^
ValueError: too many values to unpack (expected 2)
```

```
program.py > d
1   lista = ["primer", 25, [1, 2, 3]]
2   a, b, c, d = lista

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\PII\py\python-practice1> python program.py
Traceback (most recent call last):
  File "C:\PII\py\python-practice1\program.py", line 2, in <module>
    a, b, c, d = lista
    ^^^^^^^^^^
ValueError: not enough values to unpack (expected 4, got 3)
```

Para trabajar sólo con alguno valores de la lista que nos interesa, por ejemplo el primero y último elemento, no valemos del * para desempaquetar todos los valores que no nos interesa en una variable.

```
program.py > ...
1   tupla = ("Juan", 25, [1, 2, 3], 10.5)
2   primer_elemento, *_, ultimo_elemento = tupla
3
4   print(ultimo_elemento)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\PII\py\python-practice1> python program.py
10.5
```

## Otros métodos útiles con iterables

### Método extend

el método **extend( )**, a diferencia de **append( )**, itera sobre el elemento que desea agregar, es decir, involucra el argumento pasado por parámetro dentro de un ciclo, y luego agrega los valores contenidos dentro de ese parámetro, es decir, el argumento que se pasa debe ser un iterable cómo por ejemplo otra lista, una tupla, una cadena, ….

```
program.py > ...
1   palabras = []
2
3   palabras.extend("chocolate")
4
5   print(palabras)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
PS C:\PII\py\python-practice1> python program.py
['c', 'h', 'o', 'c', 'o', 'l', 'a', 't', 'e']
```

```
program.py > ...
1   palabras = []
2
3   palabras.extend(['b','a','n','a','n','a'])
4
5   print(palabras)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS C:\PII\py\python-practice1> python program.py
['b', 'a', 'n', 'a', 'n', 'a']
```

program.py > ...

```
1   palabras = []
2
3   palabras.extend(('m','a','n','z','a','n','a'))
4
5   print(palabras)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\PII\py\python-practice1> python program.py
['m', 'a', 'n', 'z', 'a', 'n', 'a']
```

## Método reverse

**reverse( )** es un método incorporado en el lenguaje de programación Python que invierte los objetos de la Lista en su lugar, es decir, no utiliza ningún espacio adicional sino que simplemente modifica la lista original.

program.py > ...

```
1   list_1 = list(range(1,6))
2   print(list_1)
3   list_1.reverse()
4   print(list_1)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\PII\py\python-practice1> python program.py
[1, 2, 3, 4, 5]
[5, 4, 3, 2, 1]
```

program.py > ...

```
1   # Dada una palabra verifique si es un palíndromo
2   palabra = "reconocer"
3   lista_palabra_reverse = list(palabra)
4   lista_palabra_reverse.reverse()
5   palabra_reverse = ''.join(lista_palabra_reverse)
6
7   if palabra_reverse == palabra:
8       print(palabra, ": Es un palíndromo")
9   else:
10      print(palabra, ": No es un palíndromo")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\PII\py\python-practice1> python program.py
reconocer : Es un palíndromo
```

# Ordenamiento y filtrado complejos de Listas

Supongamos que tenemos que ordenar por más de una condición una lista. Para ello necesitamos aplicar ordenamientos secuenciales a la lista hasta obtener el resultado deseado.
Para ello utilizaremos la función **sorted( )** y el método **sort( )** y funciones **lambda**

Cuando Python realiza un proceso de ordenamiento con **sorted( )** o **sort( )**, realiza un ordenamiento estable porque ordena la lista con el algoritmo TimSort, un algoritmo de ordenamiento muy eficiente y estable.
Un algoritmo de ordenación es estable si garantiza que no se cambia el orden relativo que mantienen inicialmente los elementos que se consideran iguales. Esto es útil para realizar ordenaciones en múltiples fases.
Esto significa que si dos elementos tienen el mismo valor o valor intermedio (clave), está garantizado que se mantendrán en el mismo orden relativo entre ellos.

Supongamos que necesitamos ordenar la siguiente lista primero por longitud del nombre de la marca y dentro de la misma longitud alfabéticamente. Para obtener el resultado deseado debemos ordenar en 2 fases. Primero ordenamos alfabéticamente y luego ordenamos por longitud obtenemos el resultado deseado.

```
12  marcas = ['gmc','audi','kia','FORD','BMW','AUDI',]
13
14  marcas.sort(key=lambda x: x.upper())
15  marcas.sort(key=lambda x: len(x))
16  print(marcas)
17
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS D:\TUP\Programacion II\Practica 2\TUP-Python-Practica2> python app.py
['BMW', 'gmc', 'kia', 'audi', 'AUDI', 'FORD']
```

Observe la diferencia de ordenar por una u otra condición únicamente.

```
12  marcas = ['gmc','audi','kia','FORD','BMW','AUDI',]
13
14  marcas.sort(key=lambda x: len(x))
15  print(marcas)
16
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS D:\TUP\Programacion II\Practica 2\TUP-Python-Practica2> python app.py
['gmc', 'kia', 'BMW', 'audi', 'FORD', 'AUDI']
```

```
12  marcas = ['gmc','audi','kia','FORD','BMW','AUDI',]
13
14  marcas.sort(key=lambda x: x.upper())
15  print(marcas)
16
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS D:\TUP\Programacion II\Practica 2\TUP-Python-Practica2> python app.py
['audi', 'AUDI', 'BMW', 'FORD', 'gmc', 'kia']
```

Supongamos que necesitamos ordenar la siguiente lista de números primero de mayor a menor y luego todos los pares primero y luego los impares, de manera que me queden primero los números pares de mayor a menor y luego los impares de mayor a menor. Puedo hacerlo ordenando en 2 fases.

```
19  numeros = [2,1,10,5,7,6,3]
20  lista_ord = sorted(numeros, reverse=True)
21  lista_ord_par = sorted(lista_ord, key=lambda x: x%2 != 0)
22  print(lista_ord_par)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS D:\TUP\Programacion II\Practica 2\TUP-Python-Practica2> python app.py
[10, 6, 2, 7, 5, 3, 1]
```

Supongamos que desea filtrar los números impares de una  lista. Podríamos hacerlo con una función lambda o con un for cómo sigue:

program.py > ...

```
1  mi_lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
2  filtrado = []
3
4  for num in mi_lista:
5      if num % 2 != 0:
6          filtrado.append(num)
7
8  print(filtrado)
```

program.py > ...

```
1  filtrado = [x for x in range(1,11) if x % 2 != 0]
2
3  print(filtrado)
```

## Bibliografía


| |
|---|
| https://www.python.org/doc/ |
| https://www.w3schools.com/python/default.asp |
| https://www.freecodecamp.org/espanol/news/tag/python/ |
| Matthes, E. (2023). Python crash course: A hands-on, project-based introduction to programming. no starch press. |
| https://es.wikipedia.org/wiki/Algoritmo_de_ordenamiento |
| https://en.wikipedia.org/wiki/Timsort |


## Versiones

| Versión | |
|---|---|
| 1.0 | Versión Inicial |
| 2.0 | Se agrega contenido a conjuntos |
| | |

## Autores

| |
|---|
| María Mercedes Valoni |
| |
| |