



INFORMATARIO

101010101010 >>> ★★★



Taller de Programación Web

Estructuras de datos



Subsecretaría de
Empleo
Chaco Gobierno de todos



Ministerio de
Producción, Industria y Empleo
Chaco Gobierno de todos



CHACO
Gobierno de todos



INTRODUCCIÓN

Hasta el momento pudimos ver lo que son las variables y la estructura de control de flujo condicional, por lo que ya podemos comenzar a no solo almacenar valores en las variables, sino que también, ya podemos comenzar a controlar el flujo del programa. Pero surge algo más que debemos controlar, y es la organización de los datos y la forma de almacenarlos.

Las estructuras de datos o también **tipos de datos compuestos** son una forma de organizar y almacenar datos para que puedan ser manipulados y utilizados de manera eficiente. Una estructura de datos se refiere a una colección de valores de datos que están organizados y almacenados de una manera particular para su uso y acceso.

La elección de una estructura de datos adecuada es fundamental para el éxito de cualquier proyecto de programación, ya que una estructura de datos inapropiada puede resultar en un código lento, ineficiente e incluso incorrecto. Por lo tanto, es importante que los programadores comprendan las diferentes estructuras de datos y su uso, para poder elegir la estructura más adecuada para cada situación y problema.

Hay cuatro tipos de estructuras de datos en Python: **listas, tuplas, conjuntos y diccionarios**.

LISTA

Las listas son una colección ordenada y mutable de elementos que pueden ser de diferentes tipos de datos. Se pueden agregar, eliminar y modificar elementos de una lista. Las listas se definen utilizando corchetes [] y los elementos de la lista se separan por comas.

```
1  #lista de números
2  numeros = [1, 2, 3, 4, 5]
3
4  #lista de strings
5  nombres = ["Juan", "Maria", "Pedro"]
6
7  #lista de varios tipos de datos
8  lista = ["Juan", "Maria", "Pedro", 1, 2, 3, True, False, 3.14]
```





Podemos observar que para crear una lista, basta solo con declarar un nombre para la lista y asignar datos, de un tipo o varios, dentro de corchetes y separados por coma.

Cada uno de estos datos se convierte en un elemento de la lista, por lo que puede ser accedido mediante un índice.

nombre de la lista	signo de asignación	corchete de apertura	elemento 0	coma de separación	elemento 1	corchete de cierre
lista	=	['Hola'	,	'Mundo']

En las listas cada elemento tiene un índice contando desde cero. Es decir, que para acceder al primer elemento de nuestra lista, debemos hacerlo desde el cero. Y la forma de acceder, para mostrar lo que tiene nuestra lista es con el uso de la función print.

```
1 #lista de números
2 numeros = [1, 2, 3, 4, 5]
3
4 #lista de strings
5 nombres = ["Juan", "Maria", "Pedro"]
6
7 #lista de varios tipos de datos
8 lista = ["Juan", "Maria", "Pedro", 1, 2, 3, True, False, 3.14]
9
10 # Acceder a un elemento de la lista
11 print(numeros[0])
12
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

1

De esta manera podemos mostrar lo que tiene nuestra lista ya sea en el elemento 0 o en el elemento al cual queramos acceder, siempre con el número de índice entre corchetes, luego del nombre de la lista.

Y si queremos mostrar todos los elementos que tiene la lista, lo hacemos sin poner un índice.



```
1 #lista de números
2 numeros = [1, 2, 3, 4, 5]
3
4 #lista de strings
5 nombres = ["Juan", "Maria", "Pedro"]
6
7 #lista de varios tipos de datos
8 lista = ["Juan", "Maria", "Pedro", 1, 2, 3, True, False, 3.14]
9
10 # Acceder a un elemento de la lista
11 print(numeros)
12 print(nombres)
13 print(lista)
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
[1, 2, 3, 4, 5]
['Juan', 'Maria', 'Pedro']
['Juan', 'Maria', 'Pedro', 1, 2, 3, True, False, 3.14]
```

Y como buen observador, asumimos que te diste cuenta que para mostrar el contenido, o todos los elementos, de la lista, se hace de la misma forma que lo hacíamos con las variables.

¿Y qué más puede contener una lista?

Puede contener otras listas adentro de la lista. Una lista adentro de otra, se convierte en un elemento por lo que si queremos acceder a la lista dentro de la lista, lo hacemos con el número de índice, como lo hacemos para acceder a cualquier elemento, pero ahora veremos una lista completa en este caso.

```
1 #lista de números
2 numeros = [1, 2, 3, 4, 5]
3
4 #lista de strings
5 nombres = ["Juan", "Maria", "Pedro"]
6
7 #lista de varios tipos de datos
8 lista = ["Juan", "Maria", "Pedro", numeros, True, False, 3.14]
9
10 # Acceder a un elemento de la lista
11 print(lista[3])
12
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
[1, 2, 3, 4, 5]
```





¿Pudiste ver como guardando una lista dentro de la lista, esta se convierte en un elemento más? por lo que accediendo mediante el índice, accedemos a todo lo que contenga esa lista.

En este caso guardamos un lista creada con anterioridad, sin embargo, se puede guardar un lista adentro de la lista, sin haberla creado antes.

```
7 #lista de varios tipos de datos
8 lista = ["Juan", "Maria", "Pedro", [6, 7, 8, 9], True, False, 3.14]
9
10 # Acceder a un elemento de la lista
11 print(lista[3])
12
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
[6, 7, 8, 9]
```

Además, podemos acceder a un rango de elementos de la lista, y esto se llama **slicing**.

```
7 #lista de varios tipos de datos
8 lista = ["Juan", "Maria", "Pedro", [6, 7, 8, 9], True, False, 3.14]
9
10 # Acceder a un rango de elementos de la lista
11 print(lista[1:4])
12
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
['Maria', 'Pedro', [6, 7, 8, 9]]
```

Como vés, esto se hace poniendo el rango al que queremos acceder (escribiendo el número de la primer posición, separado por : y, el número de la última posición a la que queremos acceder). En este caso accedimos al elemento de la posición 1 (es decir, el segundo elemento de la lista) hasta el elemento de la posición 3 (que sería el elemento 4). Volvemos a recordarte que al momento de acceder, Python cuenta desde el 0 en adelante.

También se puede dar un rango desde una posición hasta el final o, desde el principio hasta una posición.

Ejemplo: desde una posición, hasta el final de la lista

```
7 #lista de varios tipos de datos
8 lista = ["Juan", "Maria", "Pedro", [6, 7, 8, 9], True, False, 3.14]
9
10 # Acceder a un rango de elementos de la lista
11 print(lista[2:])
12
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
['Pedro', [6, 7, 8, 9], True, False, 3.14]
```

Ejemplo: desde el principio de la lista, hasta una posición

```
7 #lista de varios tipos de datos
8 lista = ["Juan", "Maria", "Pedro", [6, 7, 8, 9], True, False, 3.14]
9
10 # Acceder a un rango de elementos de la lista
11 print(lista[:3])
12
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
['Juan', 'Maria', 'Pedro']
```





Y otra forma de acceder a los elementos de la lista, es desde el final hacia adelante. Esto lo hacemos usando números negativos.

```
7  #lista de varios tipos de datos
8  lista = ["Juan", "Maria", "Pedro", [6, 7, 8, 9], True, False, 3.14]
9
10 #acceder al último elemento la lista
11 print(lista[-1])
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

3.14

O, al anteúltimo, y así sucesivamente, podemos recorrer la lista desde atrás.

```
7  #lista de varios tipos de datos
8  lista = ["Juan", "Maria", "Pedro", [6, 7, 8, 9], True, False, 3.14]
9
10 #acceder al anteúltimo elemento la lista
11 print(lista[-2])
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

False

Para modificar un elemento de la lista solo basta con asignar el nuevo valor a la posición del elemento que deseamos cambiar.

```
7  #lista de varios tipos de datos
8  lista = ["Juan", "Maria", "Pedro", [6, 7, 8, 9], True, False, 3.14]
9  lista[1] = "Ana"
10
11 #Imprimir la lista
12 print(lista)
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

['Juan', 'Ana', 'Pedro', [6, 7, 8, 9], True, False, 3.14]

Y si asignamos nuevos valores a la lista, estos nuevos valores reemplazan a todos los elementos que había antes en la lista. Y todo esto que venimos haciendo es gracias a que las listas son mutables.

```
7  #lista de varios tipos de datos
8  lista = ["Juan", "Maria", "Pedro", [6, 7, 8, 9], True, False, 3.14]
9  lista = ["Hola", "Mundo", 1, True]
10
11 #Imprimir la lista
12 print(lista)
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

['Hola', 'Mundo', 1, True]





Métodos para listas

Además, podemos hacer otras cosas con las listas, como agregar elementos, eliminarlos, etc, mediante **métodos**, la cual es una nueva palabra que estamos introduciendo al vocabulario, por lo que te vamos a dejar una pequeña explicación y diferencia con **funciones** (las cuales venimos usando desde el primer día).

Funciones

Una función es un bloque de código que se puede llamar desde cualquier parte de un programa para realizar una tarea específica. Esta puede recibir parámetros (valores que se le pasan a la función) y devolver un resultado. Por ejemplo, la función print a la cual le pasamos un parámetro y muestra por pantalla ese valor ----> `print(variable)`

Métodos

Un método es una función que está asociada con un objeto en particular. En Python, los objetos tienen métodos que les permiten realizar acciones específicas. Por ejemplo, una lista tiene un método llamado `append()`, que se utiliza para agregar elementos al final de la lista. En este caso, la lista es el objeto al que pertenece el método `append()`, y el método solo puede ser utilizado con objetos de tipo lista.

Así que podemos decir que, la principal diferencia entre una función y un método en Python, es que una función es independiente y puede ser llamada desde cualquier parte del programa (o incluso desde otros programas), mientras que un método está asociado con un objeto en particular y solo puede ser utilizado por objetos de ese tipo.

Así que explicado brevemente esto (más adelante vas a comprender aún mejor esto), vamos a ver los métodos más comunes que se puede usar con listas.

- * **append(elemento):** Agrega un elemento al final de la lista.
- * **extend(iterable):** Agrega los elementos de un iterable al final de la lista.
- * **insert(i, elemento):** Inserta un elemento en una posición específica de la lista.
- * **remove(elemento):** Elimina la primera aparición de un elemento de la lista.
- * **pop([i]):** Elimina el elemento en la posición especificada y lo devuelve. Si no se especifica una posición, elimina y devuelve el último elemento de la lista.





- * **clear()**: Elimina todos los elementos de la lista.
- * **index(elemento[, inicio[, fin]])**: Devuelve la posición de la primera aparición de un elemento en la lista. Se pueden especificar los índices de inicio y fin para hacer un slicing.
- * **count(elemento)**: Devuelve el número de veces que un elemento aparece en la lista.
- * **reverse()**: Invierte el orden de los elementos en la lista.
- * **copy()**: Devuelve una copia superficial de la lista.
- * **sort(key=None, reverse=False)**: Ordena los elementos de la lista en orden ascendente (a menos que se especifique lo contrario con el parámetro reverse=True). Se puede especificar una función de clave opcional para definir el orden de clasificación.
- * **len()**: devuelve el número de elementos en la lista.

Te damos algunos ejemplos en código de como usar alguno de estos métodos

append()

```
7  #lista de varios tipos de datos
8  lista = ["Hola", "Mundo", 1, True]
9
10 #Imprimir la lista
11 print(lista)
12
13 # Agregar un elemento a la lista
14 lista.append("otro elemento")
15
16 #Imprimir la lista
17 print(lista)
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
['Hola', 'Mundo', 1, True]
['Hola', 'Mundo', 1, True, 'otro elemento']
```





remove()

```
7  #lista de varios tipos de datos
8  lista = ["Hola", "Mundo", 1, True]
9
10 #Imprimir la lista
11 print(lista)
12
13 # Eliminar un elemento de la lista
14 lista.remove("Mundo")
15
16 #Imprimir la lista
17 print(lista)
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
[ 'Hola', 'Mundo', 1, True]
[ 'Hola', 1, True]
```

len()

```
7  #lista de varios tipos de datos
8  lista = ["Hola", "Mundo", 1, True]
9
10 #Imprimir la lista
11 print(len(lista))
12
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
4
```

Y para no aburrirte te invitamos a que practiques y utilices los demás métodos que te presentamos, e incluso te invitamos a que investigues y practiques con otros métodos que existe para listas.



Subsecretaría de
Empleo



Ministerio de
Producción, Industria y Empleo



CHACO
Gobierno de todos



TUPLA

Una tupla es una colección de elementos ordenados e inmutables. Las tuplas son similares a las listas, pero a diferencia de las listas, una vez que se crea una tupla, no se pueden agregar, eliminar o modificar elementos.

Las tuplas se pueden crear utilizando paréntesis () y los elementos se separan por comas. Incluso si una tupla tiene un solo elemento, se debe incluir una coma al final para indicar que es una tupla en lugar de un valor entre paréntesis.

Te damos un ejemplo de cómo crear una tupla:

```
1     tupla = ("manzana", "naranja", "banana")
```

Como podés observar, tiene prácticamente la estructura de una lista para crearla, pero en vez de corchetes, lleva paréntesis. Sin embargo, las tuplas pueden ser creadas sin paréntesis (siguiendo el mismo orden de separar los elementos por coma), pero por convención siempre usamos paréntesis.

```
1     tupla = ("manzana", "naranja", "banana")
2     tupla1 = "manzana", "naranja", "banana"
3
4     print(type(tupla))
5     print(type(tupla1))
6
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
<class 'tuple'>
<class 'tuple'>
```

Aquí te mostramos el ejemplo de lo que te decíamos anteriormente, sin embargo, recalcamos, por convención vamos a crear las tuplas con paréntesis.

Algo que ya usamos anteriormente en los tipos de datos simples, es la función type(), la cual también se puede usar para listas, tuplas, conjuntos y diccionarios.

Y como mencionamos antes, si queremos crear una tupla, en este caso, de un solo elemento, siempre tenemos que poner la coma al final.

```
1     tupla = ("manzana",)
2
3     print(type(tupla))
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
<class 'tuple'>
```





Una tupla es inmutable

Cuando nos referimos a esto, estamos diciendo que las tuplas no pueden ser modificadas, a diferencia de las listas, las cuales si queríamos modificar un elemento, podíamos asignar un nuevo valor ingresando el nuevo elemento mediante su índice, en las tuplas no ocurre esto. Si bien podemos mostrar por pantalla un elemento de la tupla mediante su índice (como lo hacíamos en las listas), no podemos modificar un elemento.

```
1  tupla = ("manzana", "pera", "banana")
2
3  #accediendo mediante índice
4  print(tupla[1])
5
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

pera

Aquí accedemos mediante índice a un elemento de la tupla. En este caso al segundo elemento.

```
1  tupla = ("manzana", "pera", "banana")
2
3  #intentando modificar una tupla
4  tupla[1] = ("pomelo")
5
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL Python

```
Traceback (most recent call last):
  File "c:\Users\Pc\Desktop\Info2023\estructuras_de_datos_tuplas.py", line 4, in <module>
    tupla[1] = ("pomelo")
    ~~~~~^~~
TypeError: 'tuple' object does not support item assignment
```

Aquí estamos intentando modificar la tupla, como lo hacíamos en las listas, sin embargo, vemos que nos da un error.

Y te estarás preguntando si una vez creada la tupla, ¿no se la puede modificar más? No por medio de índices, es decir, no se puede modificar un solo valor de la tupla, sin embargo, podemos reescribir todos los valores desde cero.

```
1  tupla = ("manzana", "pera", "banana")
2  print(tupla)
3
4  #escribiendo valores nuevos en la tupla
5  tupla = ("pomelo", "melón", "kiwi")
6  print(tupla)
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
('manzana', 'pera', 'banana')
('pomelo', 'melón', 'kiwi')
```

Y si bien las tuplas no tienen tantos métodos para usarse sobre ellas como las listas, te invitamos a que puedas probar algunos de los métodos que te dimos de listas y verás que algunos sirven para tuplas y otros no. También te animamos a que busques y practiques más métodos para tuplas.

Recordá que la práctica, más práctica y mucha más práctica te ayudará a ser un buen programador!





CONJUNTO

Un conjunto es una colección de elementos desordenados y únicos. Así como vimos que las listas y tuplas son estructuras ordenadas, ya que por medio de índices podemos acceder a los elementos que contienen, aquí no ocurre lo mismo. Cada elemento que contiene el conjunto, no tiene índice y no puede accederse como lo veníamos haciendo anteriormente.

Otra diferencia es que son únicos, ya que sus elementos no pueden repetirse. Esta es una función útil cuando queremos eliminar valores, o elementos, duplicados.

También son muy útiles para realizar operaciones de conjunto, como la unión, la intersección, la diferencia y la comprobación de la pertenencia de elementos.

Aquí para crearlos utilizaremos llaves {}

```
1  conjunto = {1, 5, "Hola", "Mundo", 3.14, True}
2  print(conjunto)
3  print(type(conjunto))
4
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
{1, 'Mundo', 3.14, 5, 'Hola'}
<class 'set'>
```

Como podés observar, mantiene la misma estructura básica de como veníamos creando las otras estructuras, con la diferencia de el uso de llaves. Pero, vamos a ver una de las características del conjunto, y es que son únicos.

```
1  conjunto = {1, 5, "Hola", "Mundo", "Hola", 1, 5, 3.14, True}
2  print(conjunto)
3
4
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
{1, 3.14, 5, 'Hola', 'Mundo'}
```

En este ejemplo estamos repitiendo los valores “Hola”, 1 y 5, sin embargo cuando mostramos por pantalla, estos valores duplicados no salen, a diferencia de las listas o tuplas en las que si podíamos ingresar valores duplicados. Aquí el conjunto está cumpliendo con su característica de eliminar la duplicidad de datos. Y para probar la otra característica de desordenado, te invitamos a copiar este código y ejecutarlo varias veces. Vas a poder apreciar que todas las veces que se muestren estos datos por pantalla, no van a estar en el mismo orden.





DICCIONARIO

Y aquí llegamos al último apartado de estructuras de datos. Un diccionario es una estructura de datos que permite almacenar información en pares clave-valor (key - value). Es decir, cada elemento del diccionario consta de una clave y un valor asociado a dicha clave. La clave debe ser única dentro del diccionario, mientras que los valores pueden ser de cualquier tipo de datos.

De la misma forma que con los conjuntos, se utilizan las llaves para crearlo, sin embargo, difiere un poco en la forma de hacerlo. Vamos a ver un ejemplo.

```
1 diccionario = {'Juan': 24, 'Ana': 30, 'Cristian': 28}
2 print(diccionario)
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
{'Juan': 24, 'Ana': 30, 'Cristian': 28}
```

Como se puede observar, comenzamos de la misma forma que las demás estructuras, con el nombre que va a tener, el signo de asignación, una llave de apertura y al final una llave de cierre, pero los datos se agregan de otra forma.

Aquí debe ir una clave (de cualquier valor), : (dos puntos) y un valor (de cualquier tipo), y para seguir agregando datos, los sepáramos por comas como el resto de las estructuras.

Para acceder a un dato del diccionario, se debe usar su clave y obtendremos el valor.

En este caso, podremos acceder a la edad de la persona, mediante su nombre.

```
1 diccionario = {'Juan': 24, 'Ana': 30, 'Cristian': 28}
2 print(diccionario['Juan'])
3
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
24
```



Y para modificar un valor lo hacemos de la siguiente forma:

```
1 diccionario = {'Juan': 24, 'Ana': 30, 'Cristian': 28}
2 diccionario['Ana'] = 18
3 print(diccionario)
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
{'Juan': 24, 'Ana': 18, 'Cristian': 28}
```

Accedemos a la clave y asignamos un nuevo valor.





Otra forma de crear diccionarios es usando la función **dict()**, y se puede usar de la siguiente manera.

```
1  una_lista = [('clave1', 'valor1'), ('clave2', 'valor2'), ('clave3', 'valor3')]
2  un_diccionario = dict(una_lista)
3  print(un_diccionario)
4  print(type(un_diccionario))
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
{'clave1': 'valor1', 'clave2': 'valor2', 'clave3': 'valor3'}
<class 'dict'>
```

o usando la sintaxis de argumentos de palabras clave para especificar las claves y los valores.

```
1  mi_diccionario = dict(clave1='valor1', clave2='valor2', clave3='valor3')
2  print(mi_diccionario)
3  print(type(mi_diccionario))
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
{'clave1': 'valor1', 'clave2': 'valor2', 'clave3': 'valor3'}
<class 'dict'>
```

Sin embargo, forma más común que vas a ver en la creación de diccionarios es la primera que te mostramos.

Métodos para diccionarios

Te vamos a enumerar algunos de los métodos más comunes para diccionarios, sin embargo, te invitamos a que los puedas probar y sacar tus conclusiones.

- * **clear():** elimina todos los elementos del diccionario.
- * **copy():** devuelve una copia superficial del diccionario.
- * **get():** devuelve el valor asociado a una clave dada, o un valor predeterminado si la clave no está presente.
- * **items():** devuelve una vista de lista de todos los pares clave-valor en el diccionario.
- * **keys():** devuelve una vista de lista de todas las claves en el diccionario.
- * **pop():** elimina y devuelve el valor asociado a una clave dada.
- * **popitem():** elimina y devuelve un par clave-valor aleatorio del diccionario.
- * **setdefault():** devuelve el valor asociado a una clave dada, o lo crea con un valor predeterminado si la clave no está presente.
- * **update():** actualiza el diccionario con los pares clave-valor de otro diccionario o iterable.
- * **values():** devuelve una vista de lista de todos los valores en el diccionario.





Vamos a ver un pequeño repaso

LISTA

Permite almacenar una colección ordenada y mutable de elementos, que pueden ser accedidos mediante un índice numérico.

Es la estructura que más frecuentemente vamos a utilizar debido a su versatilidad, orden y acceso. Recordá que para crearla se hace escribiendo un nombre para la lista, signo de asignación y entre corchetes los valores que vamos a guardar, separados por coma cada uno. Su nombre en inglés es **list**.

TUPLA

Permite almacenar una colección ordenada e inmutable de elementos, que también pueden ser accedidos mediante un índice numérico.

Al igual que las listas, la forma de crearlas es similar, pero en vez de usar corchetes, usamos paréntesis. Su nombre en inglés es **tuple**.

CONJUNTO

Permite almacenar una colección desordenada y mutable de elementos únicos y no indexados. Los elementos en un conjunto son únicos y no se permiten elementos duplicados. Al igual que las listas o tuplas, la forma de crearlas es similar, pero aquí usamos llaves en vez de corchetes o paréntesis. Su nombre en inglés es **set**.

DICCIONARIO

Permite almacenar una colección no ordenada, mutable e indexada de pares clave-valor. Las claves deben ser únicas en un diccionario (no debemos repetir el nombre de una clave), mientras que los valores pueden ser de cualquier tipo de datos. Los diccionarios son muy útiles cuando se necesita acceder rápidamente a un valor basado en una clave. Su nombre en inglés es **dict**.

Ahora que ya conoces estas estructuras, te volvemos a recomendar. Practicá mucho sobre ellas, sin miedo (nada se va a quemar si hay errores). Y luego de practicar con estas estructuras, ya podés comenzar a combinar lo aprendido en el material pasado con estructuras de control de flujo **condicional**. ¡Ya tenés herramientas fundamentales para hacer programas básicos!

En la próxima lección comenzaremos con:

ESTRUCTURAS DE CONTROL DE FLUJO REPETITIVAS

