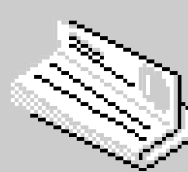
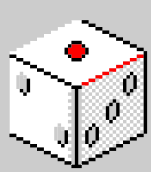
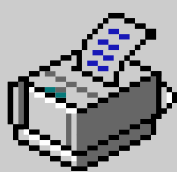
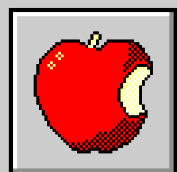


Estructuras de control en Python



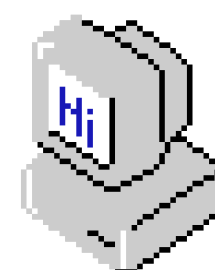
Factoría F5



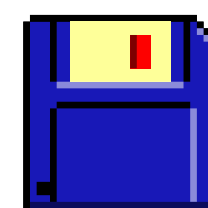
11:11AM

Grupo 3

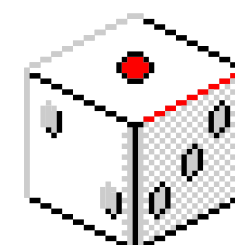
Topics Covered



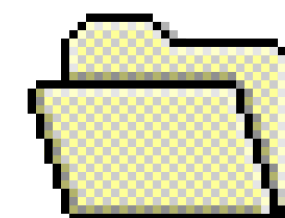
Condicionales



Bucles



Listas y tuplas



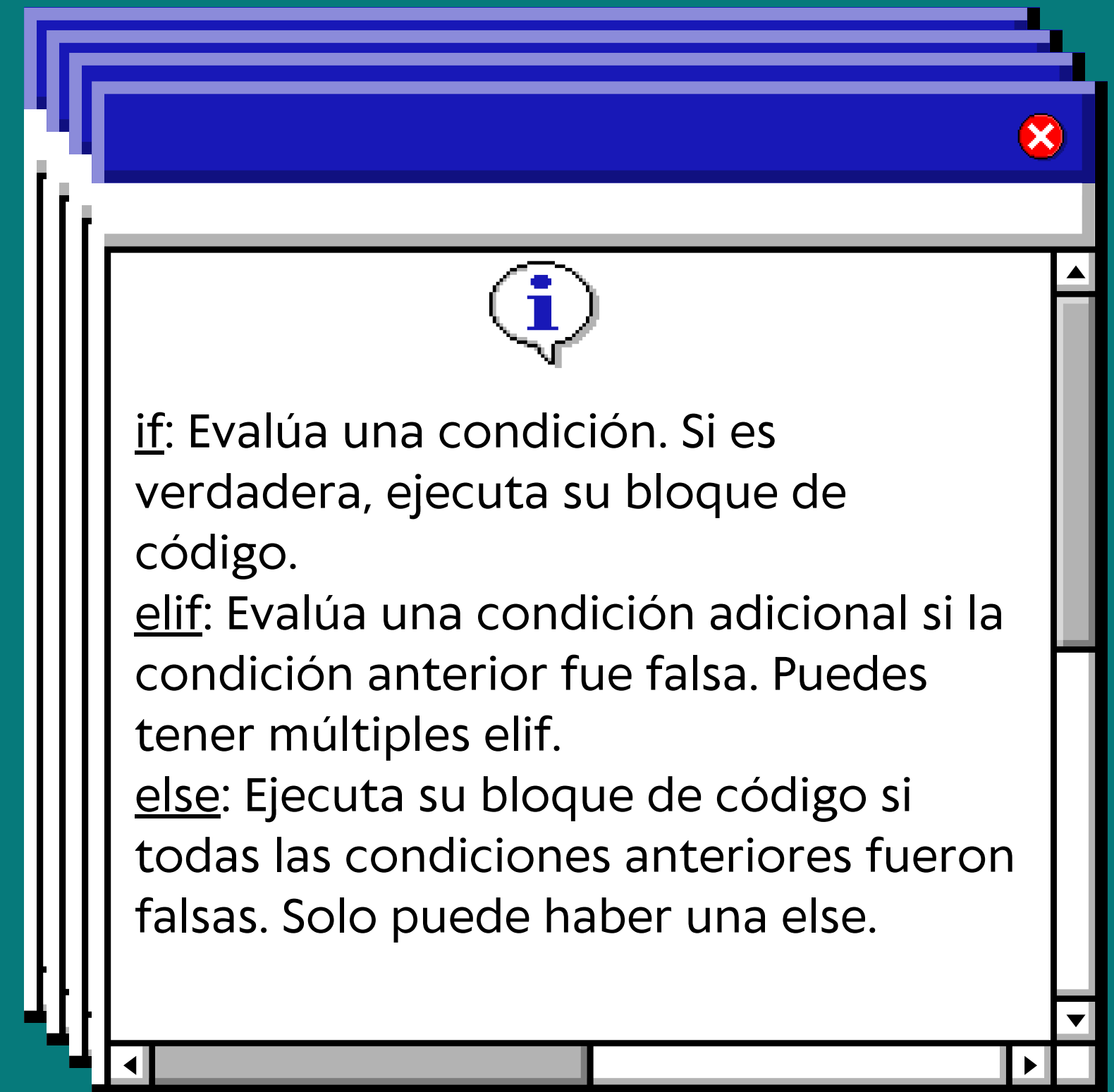
List comprehension

Start



Condicionales

- Los condicionales en Python son estructuras de control de flujo que permiten tomar decisiones y ejecutar diferentes bloques de código en función de ciertas condiciones.
- La estructura principal para los condicionales es la sentencia if, que puede combinarse con elif (abreviatura de "else if") y else para manejar múltiples condiciones.



IF - ELIF - ELSE

Bucles For y While



Los bucles nos permiten ejecutar un bloque de código repetidamente.

- El bucle **For** itera sobre una secuencia (lista, tupla, diccionario, etc.) ejecutando unas instrucciones X veces hasta que la condición o valor de salida del bucle se cumpla.
- El bucle **While** se ejecutará infinitas veces mientras se cumpla la condición que sigue a la instrucción "while". En el momento en que la condición se vuelve falsa, se saldrá fuera del bucle ejecutando el resto del código.

For - While

Bucle For

La estructura básica de un bucle **for** es la siguiente:

- for **variable** in **secuencia**:

bloque de código

if **condición**:

break # Sale del bucle si se cumple la condición

```
>>>fruits = ["apple", "cherry", "kiwi"]
```

Bucle for para recorrer cada elemento de la lista

```
>>>for fruit in fruits:  
>>>print(fruit)
```

```
>>>output
```

```
apple  
cherry  
kiwi
```

For

Bucle While

La estructura básica de un bucle **while** es la siguiente:

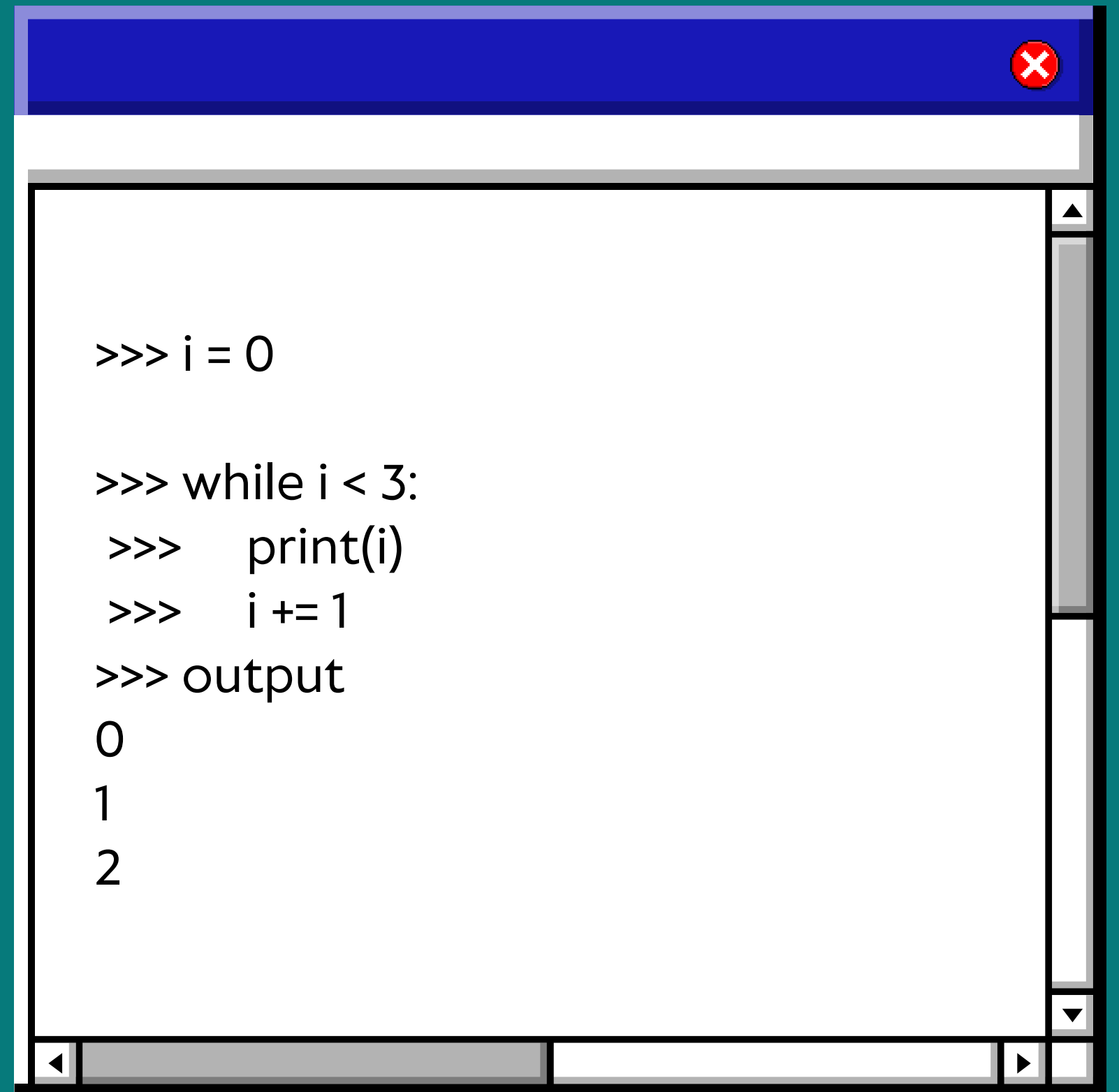
- while condición:

bloque de código

if condición_de_salida:

break

Sale del bucle si se cumple la condición de salida

A screenshot of a Python interpreter window with a blue title bar and a red close button. The window contains a code editor with the following text:

```
>>> i = 0

>>> while i < 3:
>>>     print(i)
>>>     i += 1
>>> output
0
1
2
```

The window has a scrollbar on the right side and a status bar at the bottom.

While



Diferencias entre For y While

- **Bucle for:** Es más adecuado cuando se sabe de antemano cuántas veces se quiere ejecutar el bloque de código (por ejemplo, iterar sobre los elementos de una lista).
- **Bucle while:** Es más adecuado cuando se desea que el bucle continúe ejecutándose hasta que una condición específica deje de ser verdadera (por ejemplo, esperar a que un valor cambie).

Ejemplo Completo con ambos tipos de bucles:

```
>>> fruits = ["apple", "cherry", "kiwi"]
```

```
>>> for fruit in fruits:  
    if len(fruit) >= 3:  
        print(fruit[2])
```

```
>>> fruits = ["apple", "cherry", "kiwi"]
```

```
>>> i = 0  
>>> while i < len(fruits):  
    fruit = fruits[i]  
    if len(fruit) >= 3:  
        print(fruit[2])  
    i += 1
```

For vs While

Listas

Mutable: elementos que permiten ser cambiados.

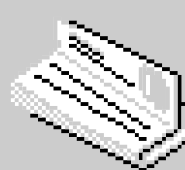
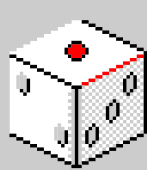
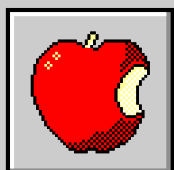
Los elementos van separados por comas y dentro de corchetes.

Las listas se pueden modificar, añadiendo o quitando elementos después de su creación.

- `Mi_lista = ["a", "b", "c", "d"]`
- `print(Mi_lista)`

- `barra_de_tareas = ["manzana", "impresora", "libreta", "dado", "planeta", "papel"]`
- `print(barra_de_tareas)`

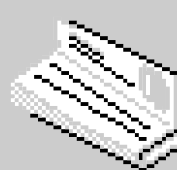
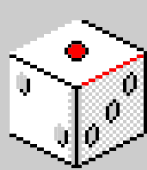
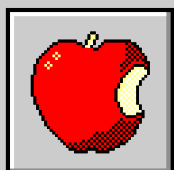
- `mi_lista = []`
- `print(mi_lista)`



Listas: Crear listas vacías y agregar elementos

.append()

Es el método que se usa para añadir elementos al final de la lista



Listas



.list()

Este método permite crear una lista cuando se quiere convertir otro tipo de datos a listas, ya sea una cadena o un rango

```
# Crear una lista a partir de una cadena
```

```
>>>cadena = "Hola Mundo"
```

```
>>>lista_de_cadena = list(cadena)
```

```
>>>print(lista_de_cadena)
```

```
>>>Output
```

```
['H', 'o', 'l', 'a', ' ', 'M', 'u', 'n', 'd', 'o']
```

```
# Crear una lista a partir de un rango
```

```
>>>rango = range(5)
```

```
>>>lista_de_rango = list(rango)
```

```
>>>print(lista_de_rango)
```

```
>>>Output
```

```
[0, 1, 2, 3, 4]
```

Listas

.extend(iterable)

Extiende la lista agregándole todos los ítems del iterable.

.insert(posicion, valor)

Permite insertar elementos a la lista. Ese elemento se insertará en la posición que indiquemos en el primer argumento.

En el segundo argumento incluiremos el valor a insertar.

```
>>>lista = ["uno", "dos", "tres"]
>>>lista2 = ["cuatro", "cinco", "seis"]
>>>lista.extend(lista2)
>>>print(lista)
```

```
>>>Output
['uno', 'dos', 'tres', 'cuatro', 'cinco', 'seis']
```

```
lista.insert(1, 'diez')
print(lista)
```

```
>>>Output
['uno', 'diez', 'dos', 'tres']
```

Manipulación de Listas

- Segmentación de listas ----->

Una segmentación usa corchetes pero en lugar de un solo elemento, tiene los índices inicial y final
ejemplo

- Combinación de listas ----->

Para unir dos listas, debe usar el otro operador (+) con dos listas para devolver una nueva.

```
>>>flores = ["Jazmin", "Rosa", "Orquidea",  
"Margarita"]  
>>>flores_before_Orquidea = flores[0:2:4]  
>>>print(flores_before_Orquidea)  
  
>>>output  
['Jazmin']  
  
>>>lista = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  
# Segmentar desde el índice 8 hasta el 2 (no  
inclusive) con un paso de -1  
>>>sub_lista = lista[8:2:-1]  
>>>print(sub_lista)  
  
>>>Output  
[8, 7, 6, 5, 4, 3]
```

```
lunas_de_jupiter = [ "Io", "Europa",  
"Ganimedes", "Calisto", "Amaltea",  
"Himalia", "Elara", "Pasifae", "Sinope",  
"Lysithea" ]
```

```
lunas_de_saturno = [ "Mimas",  
"Encélado", "Tetis", "Dione", "Rhea",  
"Titán" ]
```

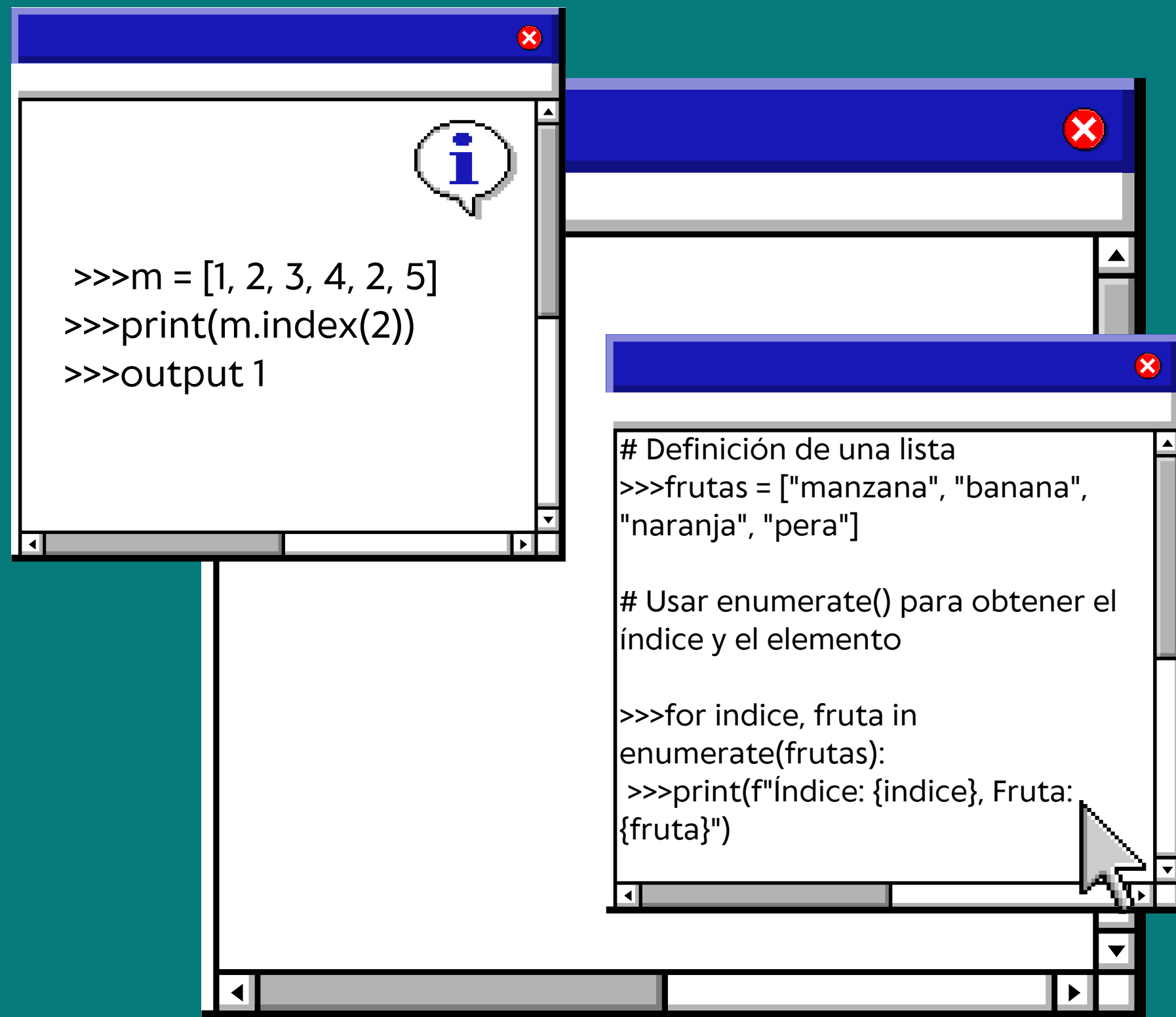
```
lunas_de_jupiter_y_saturno =  
[lunas_de_jupiter + lunas_de_saturno]  
print(lunas_de_jupiter_y_saturno)
```

Acceso a las Listas

Índices

Para acceder a un elemento de una lista se utilizan los índices. Los índices comienzan **desde cero** con el primer elemento. Así, si en una lista hay 10 elementos, el índice del último será -1 o 9, -2 sería la posición 8.

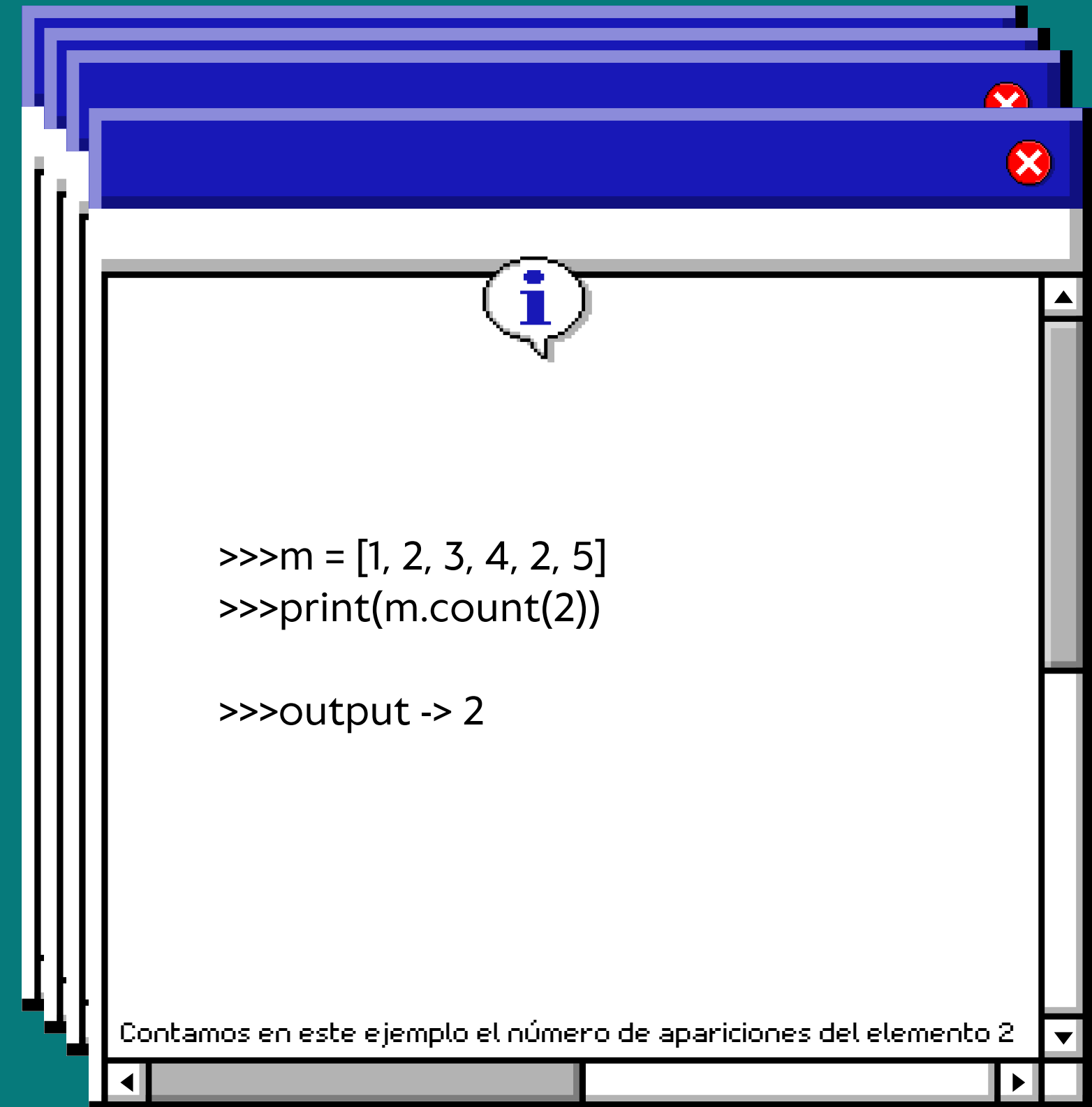
bucle for list Python – Recorrer una lista



Listas

list.count(x)

Retorna el número de veces que x aparece en la lista.



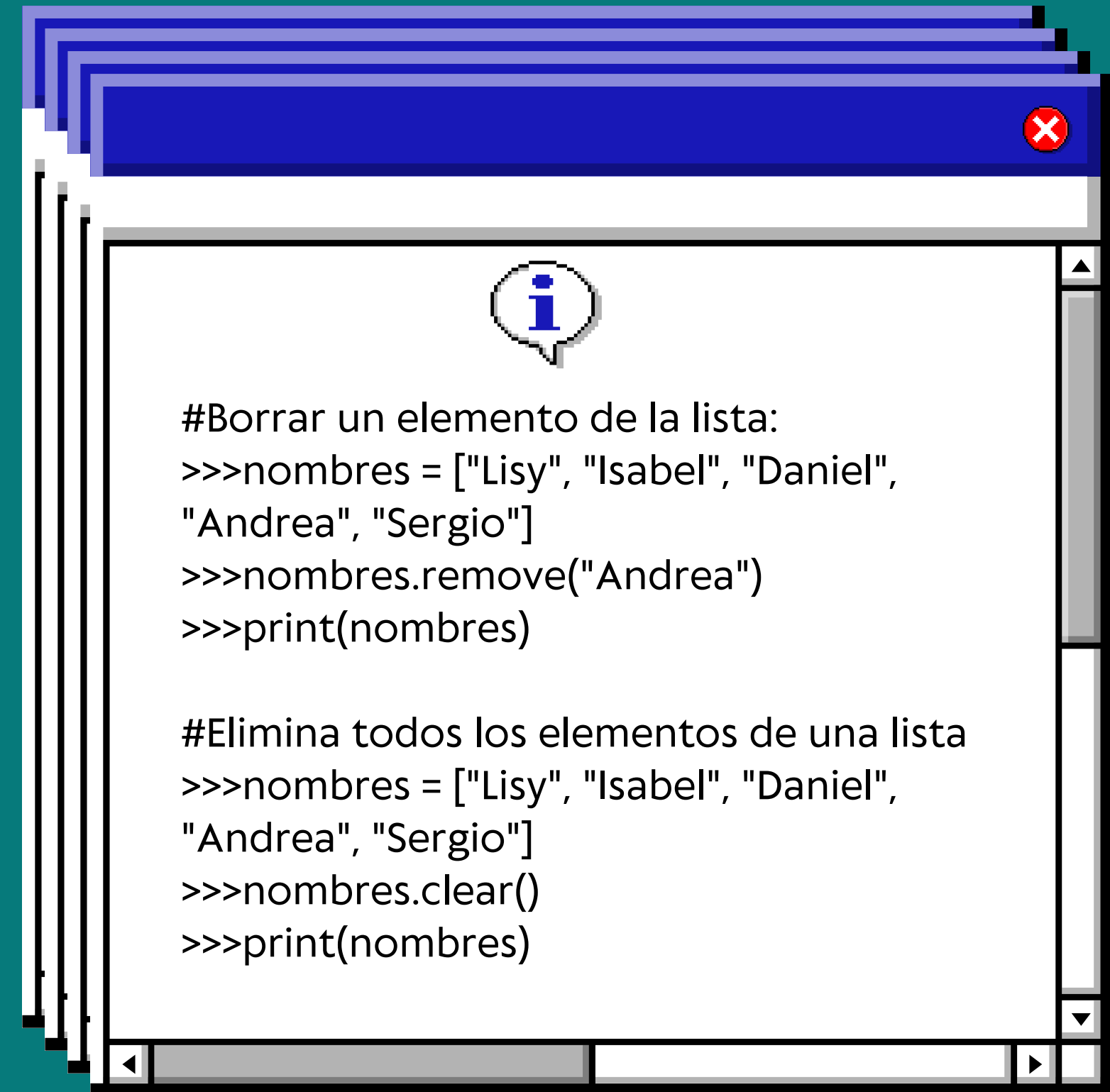
Listas

- **list.remove(x)**

Quita el primer ítem de la lista cuyo valor sea x. Lanza un ValueError si no existe tal ítem.

- **list.clear()**

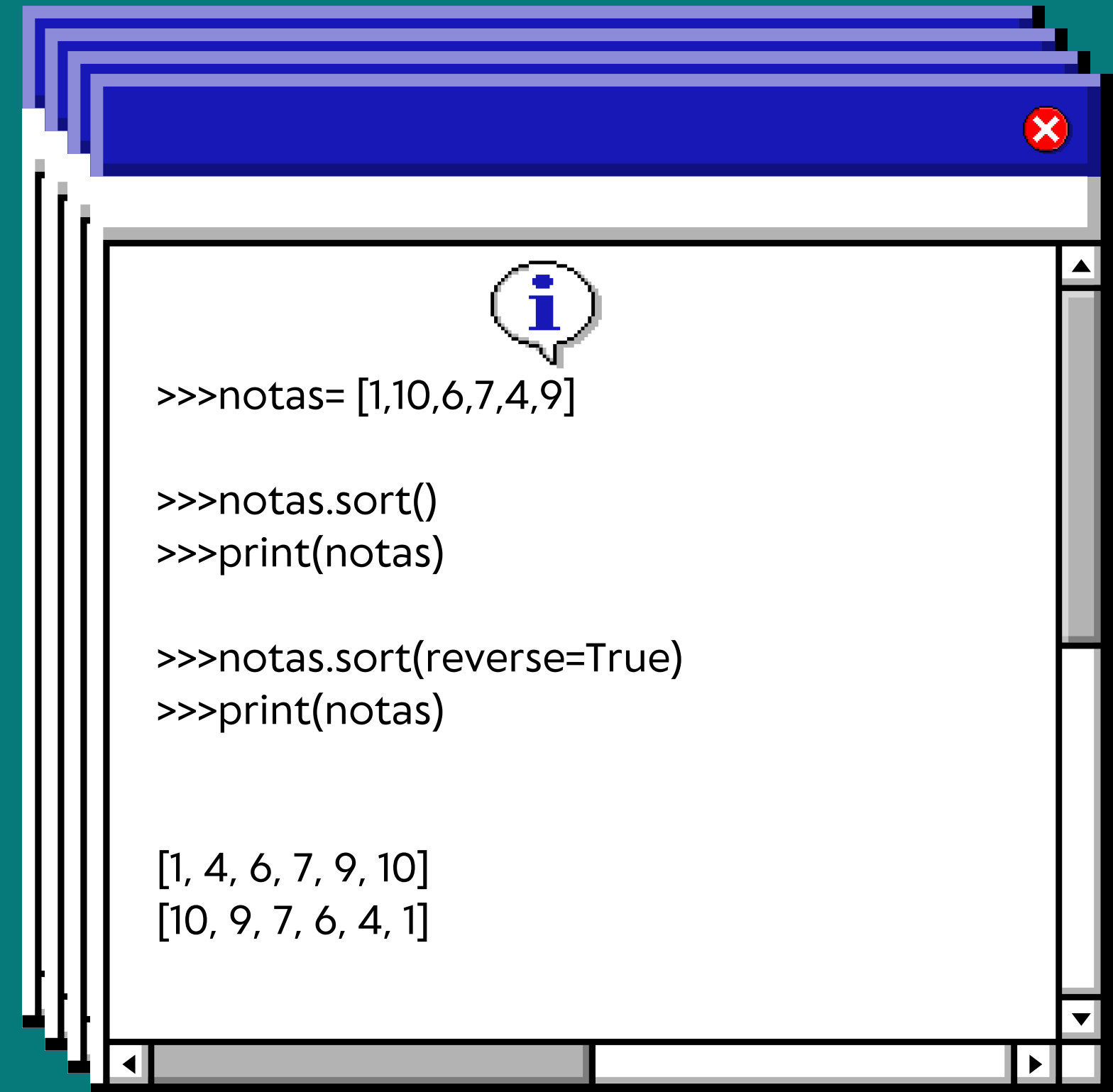
Elimina todos los elementos de la lista. Equivalente a del a[].



Listas

- **list.sort()**

Permite ordenar la lista de forma ascendente o descendente, numéricamente o alfabéticamente.



Listas

- **in**

Podemos preguntar si un elemento existe en la lista.

- **reverse()**

El método reverse() invierte el orden de la lista.



```
#Preguntamos si el elemento Andrea
existe en la lista
```

```
>>>nombres = ["Pepe", "Luis", "Alex",
"Andrea", "Rosa", "Sara"]
>>>print("Andrea" in nombres)
Output -> True
```

```
>>>numeros = [1, 2, 3]
>>>numeros.reverse()
>>>print(numeros)
```

Output

```
>>>[3, 2, 1]
```

Tuplas

Inmutable: No se puede modificar una vez creado. Son estáticas.

Los elementos van separados por coma y dentro de paréntesis.

Con las tuplas podemos realizar todas las operaciones que vimos con las listas, salvo las que conlleven una modificación «in-situ» de la misma.

Ventajas:

- Ocupan menos espacio en memoria.
- Existe protección frente a cambios indeseados.
- Las tuplas se pueden usar como claves de diccionarios (son hashables).

También pueden declararse sin (), separando por , todos sus elementos.

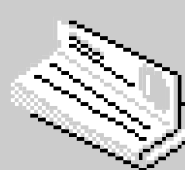
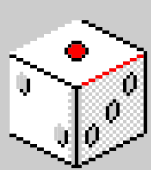
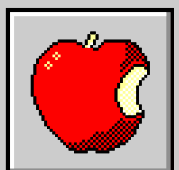
Ejemplo de tupla:

```
>>>tupla_compra = ("leche", "cacao",  
"avellanas", "azúcar")
```

Ejemplo de tupla con subtuplas:

```
>>>tupla_compra_2 = (  
("manzanas", 6), ("plátanos", 12), ("leche", 2),  
("pan", 1), ("huevos", 12), ("arroz", 1), ("pollo",  
2), ("tomates", 8), ("pasta", 3)  
)
```

```
tupla = 1, 2, 3  
print(type(tupla)) #<class 'tuple'>  
print(tupla)      #(1, 2, 3)
```



Tuplas operaciones

Podemos "sumar" dos tuplas (con lo que las concatenamos):

Podemos multiplicar una tupla por un número entero, con lo que concatenamos la tupla consigo misma tantas veces como indique el número.

También podemos comparar tuplas (lo que se realiza elemento por elemento comenzando por los de índice 0).

```
>>>m = ("a", "b", "c")  
      n = ("d", "e", "f")  
      m + n
```

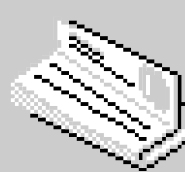
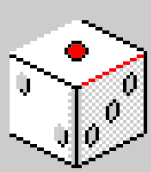
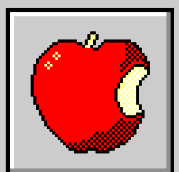
```
>>>Output  
( 'a', 'b', 'c', 'd', 'e', 'f')
```

```
>>>m = ("a", "b", "c")  
      m * 2
```

```
>>>Output  
( 'a', 'b', 'c', 'a', 'b', 'c')
```

```
>>>m = (1, 2, 3)  
>>>n = (1, 3, 4)  
      m < n
```

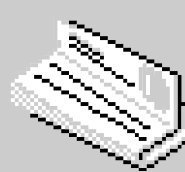
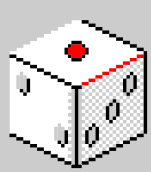
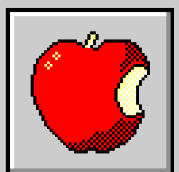
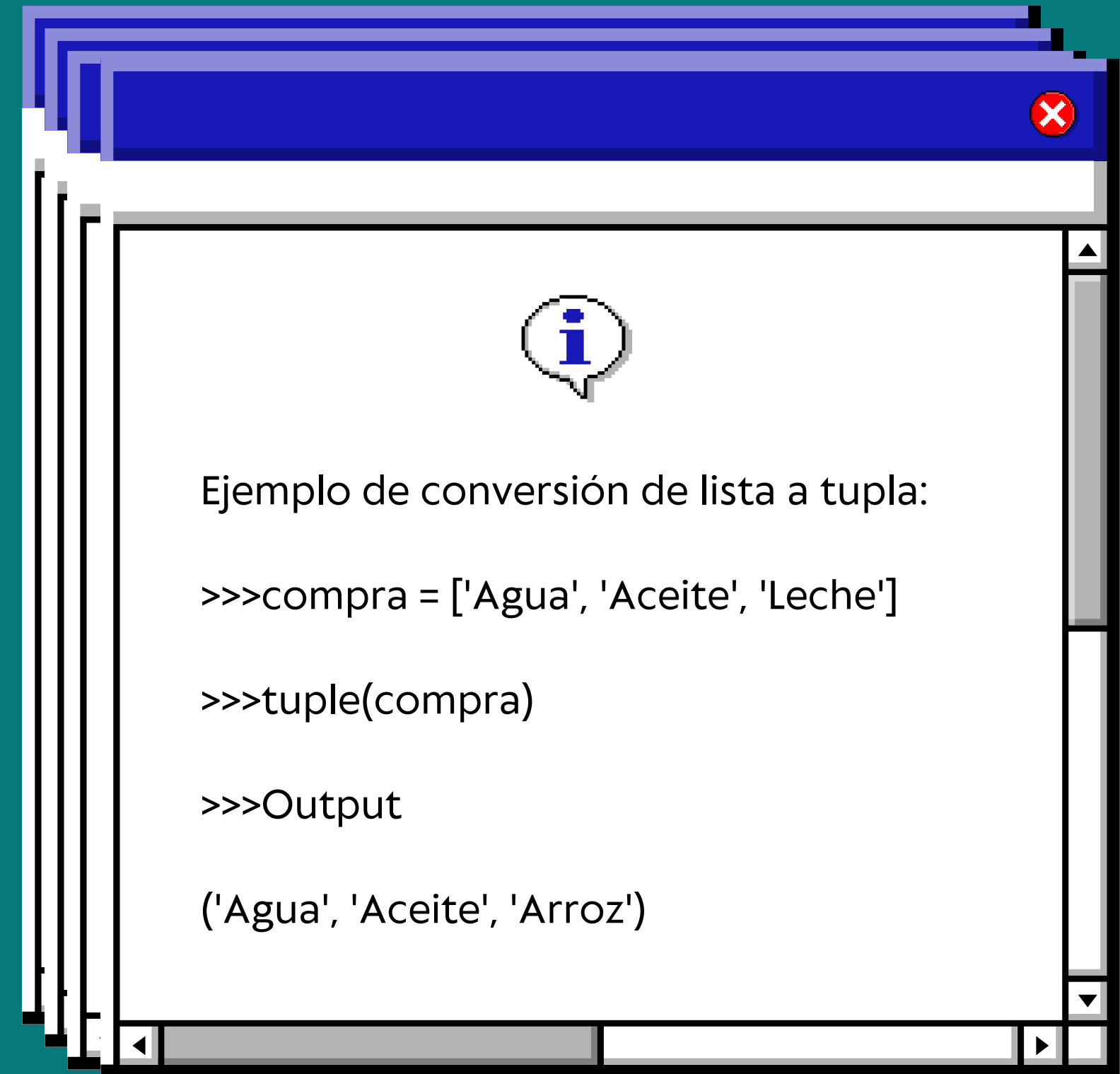
```
>>>Output  
True
```



Conversión de Listas a Tuplas

tuple()

podemos usar la función `tuple()` para convertir una lista en una tupla.

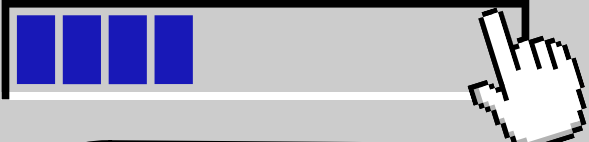


Métodos con listas y tuplas

Métodos	Listas	Tuplas
count()	x	x
index()	x	x
tuple()	x	
sort()	x	
len()	x	x
remove()	x	
append()	x	
clear()	x	



Diferencias entre Listas y Tuplas



Listas dentro de tuplas: permiten incluir una lista como uno de los elementos dentro de una tupla. Esto permite modificar los elementos de la lista contenida (agregar, eliminar o cambiar elementos).

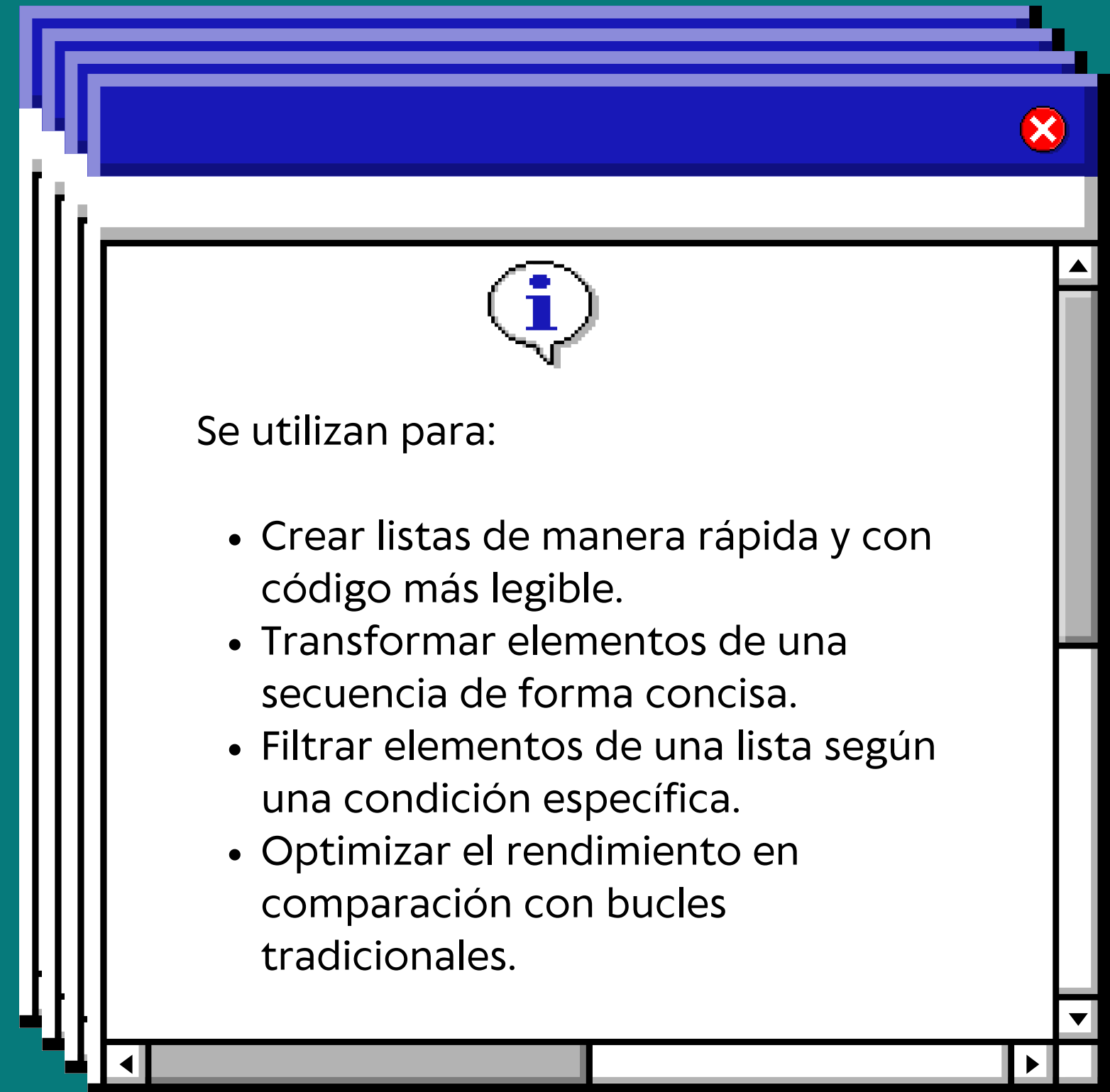
Tuplas dentro de listas: garantizan que los elementos de la tupla permanezcan inmutables. Son útiles cuando necesitas almacenar información constante o protegerla contra cambios accidentales.

La elección entre estas estructuras depende de si buscas flexibilidad o integridad de datos.

	Creacion	Mutabilidad	Elementos	Uso
Listas []	Se crea con corchetes y los elementos van separados por comas,si son numeros no es necesario que vayan entre " ".	Sí	Se pueden añadir o quitar durante y después de crear la lista	<p>Bucles</p> <ul style="list-style-type: none">• Manipulación de datos que necesita ser cambiada.• Orden de elementos que puede variar.• Necesidad de operaciones como agregar, eliminar o reordenar elementos.
Tuplas ()	Se declaran con parentesis	No	Solo se pueden añadir al crear la tupla por primera vez	<ul style="list-style-type: none">• Datos que no deberían cambiar a lo largo del tiempo.• Uso como claves en diccionarios.• Situaciones donde la inmutabilidad es una ventaja• Agrupación de elementos heterogéneos.

Crear Listas por comprensión

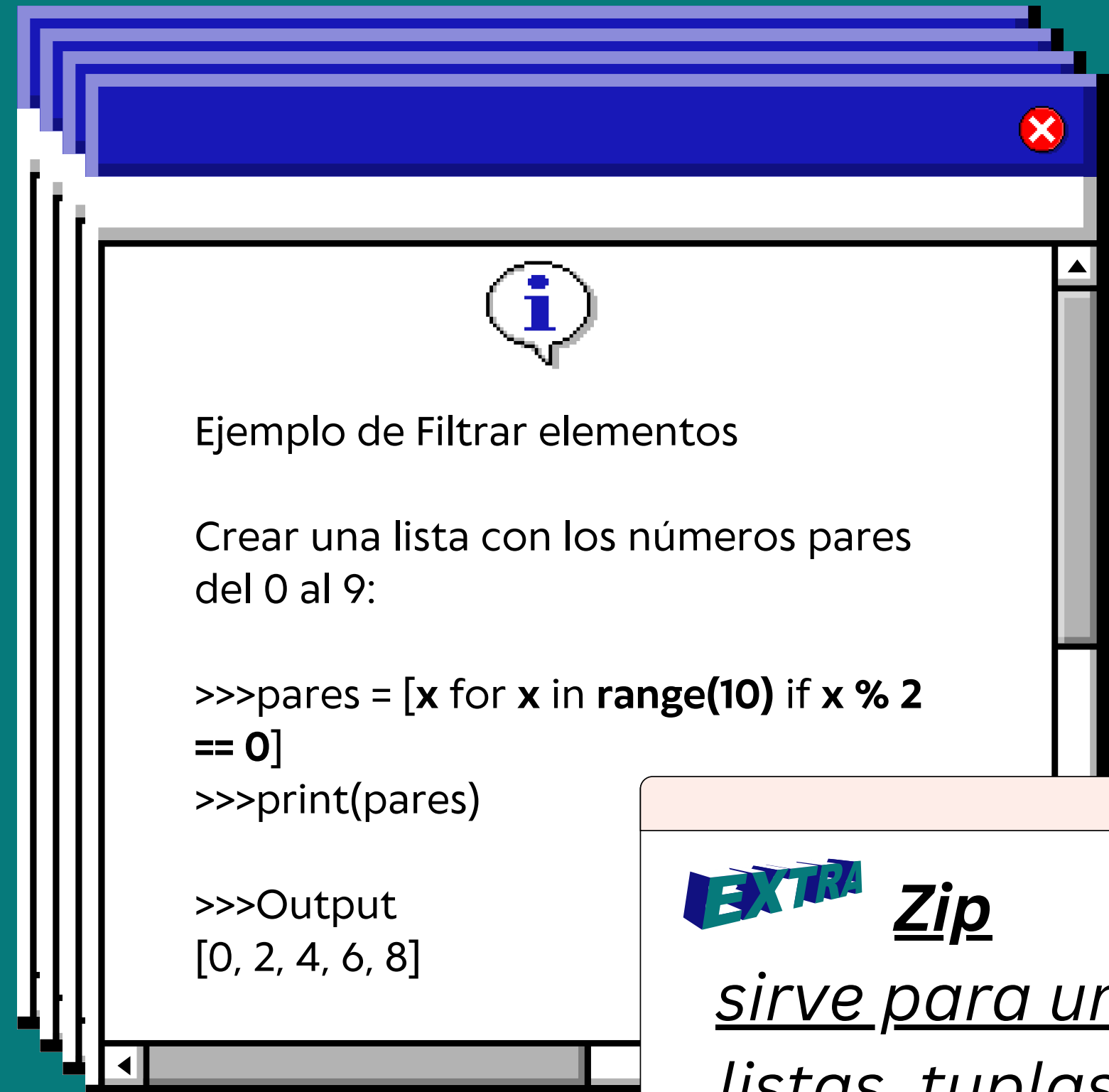
Es una técnica concisa para crear y manipular listas aplicando una expresión a cada elemento de una secuencia y, opcionalmente, filtrando elementos con una condición.



Sintaxis básica de Listas por comprensión

[**expression** for **item** in **iterable** if **condition**]

- **expression**: Cualquier expresión válida en Python, que puede incluir operaciones, llamadas a funciones, etc.
- **item**: El nombre de la variable que toma el valor de cada elemento en el iterable.
- **iterable**: Cualquier objeto que se puede iterar, como una lista, un rango, etc.
- **condition** (opcional): Una expresión booleana que filtra los elementos. Solo los elementos para los cuales esta condición es verdadera serán incluidos en la nueva lista.



EXTRA

Zip

sirve para unir
listas, tuplas y
sets.

