Listas

Algoritmos y Programación Básica (CC2005) - 2026

Listas

Semestre 02, 2026

¿Qué es una lista?

Hasta ahora, cada variable guarda un solo valor.

```
nota1 = 85
nota2 = 90
nota3 = 78
```

¿Qué pasa si hay 30 estudiantes?

¿O 1000 productos?

La solución: una lista

Una lista es una colección ordenada de valores almacenados en una sola variable.

```
notas = [85, 90, 78]
frutas = ["manzana", "naranja", "uva"]
datos = [True, 42, "Python", 3.14]
```

- Los elementos van entre corchetes `[ ]`, separados por comas.

- Pueden ser de cualquier tipo.
- El orden importa — los elementos mantienen su posición.

Ventajas

| Sin lista | Con lista | | --- | --- | | `nota1 = 85` | `notas = [85, 90, 78]` | | `nota2 = 90` | Una sola variable | | `nota3 = 78` | Se puede recorrer con un ciclo |

Crear una lista vacía

```
lista = []
```

Útil cuando se construye la lista poco a poco durante el programa.

Longitud de una lista

La función `len()` devuelve la cantidad de elementos.

```
frutas = ["manzana", "naranja", "uva", "mango"]
print(len(frutas))
```

Resultado:

```
4
```

Índices

Cada elemento de una lista tiene una posición llamada **índice**.

Los índices empiezan en `0`.

Ejemplo visual

```
frutas = ["manzana", "naranja", "uva", "mango"]
```

```
 manzana   naranja   uva    mango
    0         1       2       3
```

Acceder a un elemento

```
frutas = ["manzana", "naranja", "uva", "mango"]

print(frutas[0])   # manzana
print(frutas[1])   # naranja
print(frutas[3])   # mango
```

Índices negativos

Los índices negativos cuentan desde el final.

```
 manzana   naranja   uva    mango
   -4        -3      -2      -1
```

```
frutas = ["manzana", "naranja", "uva", "mango"]

print(frutas[-1])   # mango
print(frutas[-2])   # uva
print(frutas[-4])   # manzana
```

Modificar un elemento

A diferencia de los strings, las listas **sí se pueden modificar**.

```
frutas = ["manzana", "naranja", "uva"]

frutas[1] = "limón"
```

```
print(frutas)   # ["manzana", "limón", "uva"]
```

Error común: índice fuera de rango

```
frutas = ["manzana", "naranja", "uva"]

print(frutas[3])   # ERROR: IndexError
```

El índice máximo es `len(lista) - 1`.

Para una lista de 3 elementos, el índice máximo es `2`.

Slicing

Se puede extraer una porción de la lista.

Sintaxis: `lista[inicio:fin]`

El elemento en la posición `fin` no se incluye.

Ejemplo

```
numeros = [10, 20, 30, 40, 50]

print(numeros[1:4])    # [20, 30, 40]
print(numeros[:3])     # [10, 20, 30]
print(numeros[2:])     # [30, 40, 50]
print(numeros[:])      # [10, 20, 30, 40, 50]
```

Slice con paso

```
numeros = [10, 20, 30, 40, 50, 60]

print(numeros[::2])    # [10, 30, 50]   — cada 2 elementos
print(numeros[::-1])   # [60, 50, 40, 30, 20, 10]   — al revés
```

Métodos de listas

Un método es una función que pertenece a un tipo de dato.

Se llama con un punto: `lista.metodo()`

append — agregar al final

```
frutas = ["manzana", "naranja"]

frutas.append("uva")

print(frutas)   # ["manzana", "naranja", "uva"]
```

`append` siempre agrega el elemento al **final** de la lista.

insert — agregar en una posición

```
frutas = ["manzana", "naranja", "uva"]

frutas.insert(1, "limón")

print(frutas)   # ["manzana", "limón", "naranja", "uva"]
```

`insert(índice, valor)` — inserta el valor en la posición indicada.

Los elementos siguientes se desplazan hacia la derecha.

remove — eliminar por valor

```
frutas = ["manzana", "naranja", "uva"]

frutas.remove("naranja")

print(frutas)   # ["manzana", "uva"]
```

Elimina la **primera aparición** del valor.

Si el valor no existe, produce un error.

pop — eliminar por índice

```
frutas = ["manzana", "naranja", "uva"]

eliminado = frutas.pop(1)

print(eliminado)   # naranja
print(frutas)      # ["manzana", "uva"]
```

`pop()` sin argumento elimina el **último** elemento.

A diferencia de `remove`, devuelve el elemento eliminado.

remove vs pop

| Método | ¿Cómo elimina? | ¿Devuelve algo? | | --- | --- | --- | | `remove(valor)` | Por valor | No | | `pop(índice)` | Por índice | Sí, el elemento eliminado | | `pop()` | Último elemento | Sí, el elemento eliminado |

sort — ordenar

```
numeros = [5, 2, 8, 1, 9, 3]

numeros.sort()
```

```
print(numeros)   # [1, 2, 3, 5, 8, 9]
```

```
frutas = ["uva", "manzana", "naranja"]

frutas.sort()

print(frutas)   # ["manzana", "naranja", "uva"]
```

`sort()` modifica la lista original en lugar de crear una nueva.

sort en orden inverso

```
numeros = [5, 2, 8, 1, 9]

numeros.sort(reverse=True)

print(numeros)   # [9, 8, 5, 2, 1]
```

reverse — invertir

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

numeros.reverse()

print(numeros)   # [5, 4, 3, 2, 1]
```

`reverse()` invierte el orden sin ordenar.

`sort(reverse=True)` ordena de mayor a menor.

index — buscar posición

```
frutas = ["manzana", "naranja", "uva"]
```

```
pos = frutas.index("naranja")
print(pos)   # 1
```

Devuelve el índice de la **primera aparición**.

Si el valor no existe, produce un error.

count — contar apariciones

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

print(numeros.count(2))   # 3
print(numeros.count(5))   # 0
```

extend — agregar otra lista

```
lista1 = [1, 2, 3]
lista2 = [4, 5, 6]

lista1.extend(lista2)

print(lista1)   # [1, 2, 3, 4, 5, 6]
```

A diferencia de `append`, `extend` agrega todos los elementos de la segunda lista.

```
lista = [1, 2, 3]

lista.append([4, 5])     # agrega UNA lista: [1, 2, 3, [4, 5]]
lista.extend([4, 5])     # agrega los elementos: [1, 2, 3, 4, 5]
```

clear — vaciar la lista

```
frutas = ["manzana", "naranja", "uva"]
```

```
frutas.clear()

print(frutas)   # []
```

Tabla de métodos

| Método | ¿Qué hace? | Modifica la lista? | | --- | --- | --- | | `append(v)` | Agrega `v` al final | Sí | | `insert(i, v)` | Inserta `v` en posición `i` | Sí | | `remove(v)` | Elimina primera aparición de `v` | Sí | | `pop(i)` | Elimina y devuelve elemento en `i` | Sí | | `sort()` | Ordena de menor a mayor | Sí | | `reverse()` | Invierte el orden | Sí | | `index(v)` | Devuelve índice de `v` | No | | `count(v)` | Cuenta cuántas veces aparece `v` | No | | `extend(lista)` | Agrega todos los elementos de otra lista | Sí | | `clear()` | Elimina todos los elementos | Sí |

Iterar una lista con for

El ciclo `for` puede recorrer directamente los elementos de una lista.

Recorrer los elementos

```
frutas = ["manzana", "naranja", "uva", "mango"]

for fruta in frutas:
    print(fruta)
```

Resultado:

```
manzana
naranja
uva
mango
```

- `fruta` toma automáticamente el valor de cada elemento.
- No se necesita `range()` ni índice.

Operar sobre cada elemento

```
notas = [85, 90, 78, 92, 70]
total = 0

for nota in notas:
    total = total + nota

promedio = total / len(notas)
print("Promedio:", promedio)
```

Resultado:

```
Promedio: 83.0
```

Iterar con índice: range y len

Cuando también se necesita el índice durante el recorrido.

```
frutas = ["manzana", "naranja", "uva"]

for i in range(len(frutas)):
    print(i, frutas[i])
```

Resultado:

```
0 manzana
1 naranja
2 uva
```

Ejemplo: buscar el mayor

```
numeros = [5, 2, 8, 1, 9, 3]
mayor = numeros[0]

for numero in numeros:
    if numero > mayor:
        mayor = numero

print("El mayor es:", mayor)
```

Resultado:

```
El mayor es: 9
```

Tabla de ejecución

| Iteración | numero | mayor | | --- | --- | --- | | inicio | — | 5 | | 1 | 5 | 5 | | 2 | 2 | 5 | | 3 | 8 | 8 | | 4 | 1 | 8 | | 5 | 9 | 9 | | 6 | 3 | 9 |

Iterar una lista con while

El ciclo `while` también puede recorrer una lista usando un índice manual.

Sintaxis básica

```
frutas = ["manzana", "naranja", "uva"]
i = 0

while i < len(frutas):
    print(frutas[i])
    i = i + 1
```

Resultado:

```
manzana
naranja
uva
```

- `i` es el índice que se actualiza manualmente.
- La condición `i < len(frutas)` evita el error de índice fuera de rango.

Tabla de ejecución

| i | ¿i < 3? | frutas[i] | i después | | --- | --- | --- | --- | | 0 | Sí | manzana | 1 | | 1 | Sí | naranja | 2 | | 2 | Sí | uva | 3 | | 3 | No | — termina — | — |

Recorrer hacia atrás con while

```
numeros = [10, 20, 30, 40, 50]
i = len(numeros) - 1

while i >= 0:
    print(numeros[i])
    i = i - 1
```

Resultado:

```
50
40
30
20
10
```

Se empieza desde el último índice y se decrementa hasta llegar a 0.

for vs while para listas

| Criterio | for | while | | --- | --- | --- | | Recorrer todos los elementos | ✅ Más simple | ✅ Funciona | | Control manual del índice | No directo | ✅ Total control | | Recorrer hacia atrás | Con `[::-1]` | ✅ Natural | | Riesgo de ciclo infinito | No | Sí, si se olvida `i = i + 1` | | Recomendado para listas | ✅ Primera opción | Cuando se necesita el índice |

Ejemplo 1: Lista de calificaciones

Ingresar 5 notas y mostrar cuáles aprobaron (≥ 61).

```
notas = []

for i in range(1, 6):
    nota = float(input(f"Nota {i}: "))
    notas.append(nota)

print("\nResultados:")
for i in range(len(notas)):
    if notas[i] >= 61:
        print(f"Nota {i+1}: {notas[i]} — Aprobado")
    else:
        print(f"Nota {i+1}: {notas[i]} — Reprobado")
```

Resultado con entrada 85, 55, 90, 45, 70

```
Nota 1: 85.0 — Aprobado
Nota 2: 55.0 — Reprobado
Nota 3: 90.0 — Aprobado
Nota 4: 45.0 — Reprobado
Nota 5: 70.0 — Aprobado
```

Ejemplo 2: Inventario de productos

```
productos = ["Laptop", "Monitor", "Teclado", "Mouse"]
precios = [5000, 1200, 350, 150]

print("Inventario:")

i = 0
while i < len(productos):
    print(f"{productos[i]}: Q{precios[i]}")
    i = i + 1

total = 0
for precio in precios:
    total = total + precio

print(f"\nValor total del inventario: Q{total}")
```

Resultado

```
Inventario:
Laptop: Q5000
Monitor: Q1200
Teclado: Q350
Mouse: Q150

Valor total del inventario: Q6700
```

Ejemplo 3: Filtrar elementos

Separar números pares e impares de una lista.

```
numeros = [3, 8, 15, 4, 22, 7, 10]

pares = []
impares = []
```

```
for numero in numeros:
    if numero % 2 == 0:
        pares.append(numero)
    else:
        impares.append(numero)

print("Pares:", pares)
print("Impares:", impares)
```

Resultado:

```
Pares: [8, 4, 22, 10]
Impares: [3, 15, 7]
```

Errores comunes

Error 1: índice fuera de rango

```
frutas = ["manzana", "naranja", "uva"]

print(frutas[3])   # ERROR: IndexError
```

La lista tiene 3 elementos — los índices válidos son 0, 1 y 2.

Error 2: olvidar actualizar el índice en while

```
frutas = ["manzana", "naranja", "uva"]
i = 0

while i < len(frutas):
    print(frutas[i])
    # ⚠ Falta: i = i + 1 → ciclo infinito
```

Sin `i = i + 1`, el índice nunca avanza y el ciclo no termina.

Error 3: append dentro de append

```
lista = []
lista.append(1)
lista.append(2)
lista.append([3, 4])   # agrega una lista dentro de la lista

print(lista)   # [1, 2, [3, 4]]
```

Si se quieren agregar los elementos individuales, usar `extend`.

```
lista.extend([3, 4])   # [1, 2, 3, 4]
```

Error 4: modificar una lista mientras se itera

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

for numero in numeros:
    numeros.remove(numero)   # ⚠ comportamiento inesperado

print(numeros)   # [2, 4] — no se eliminaron todos
```

No modificar la lista durante el recorrido con `for`.

La solución: trabajar sobre una copia o construir una nueva lista.

Error 5: confundir remove con pop

```
frutas = ["manzana", "naranja", "uva"]

frutas.remove(1)    # ERROR — remove recibe el VALOR, no el índice
frutas.pop("uva")   # ERROR — pop recibe el ÍNDICE, no el valor
```

`remove` → recibe el valor. `pop` → recibe el índice.

Buenas prácticas

- Usar `for fruta in frutas` cuando solo se necesitan los valores.
- Usar `for i in range(len(lista))` cuando también se necesita el índice.
- Usar `while` cuando el recorrido no es estrictamente secuencial.
- Verificar que el índice es válido antes de acceder: `0 <= i < len(lista)`.
- Usar nombres descriptivos: `for estudiante in estudiantes` es más claro que `for x in lista`.

Antes de escribir el ciclo

Preguntarse:

- ¿Solo necesito los valores? → `for elemento in lista`
- ¿Necesito el índice también? → `for i in range(len(lista))`
- ¿El recorrido no es lineal o necesito saltar índices? → `while`