

## ¿Qué es una lista?

- Una lista es una secuencia de elementos.
- Las listas son a Python lo que los arreglos son para otros lenguajes.
- Son mucho más flexibles que los vectores tradicionales.



- Pueden contener elementos homogéneos (del mismo tipo) o combinar distintos tipos de dato.
- Por convención, en las listas se guardan elementos homogéneos.

@ Lic. Ricarde Thempsen

#### Creación de listas

 Las listas se crean al asignar a una variable una secuencia de elementos, encerrados entre corchetes y separados por comas.

> numeros = [1, 2, 3, 4, 5] dias = ["lunes", "martes", "jueves"]



 Los corchetes pueden estar juntos, lo que permite crear una lista vacía.

elementos = []

 Una lista puede contener, a su vez, otras listas:

sublista = [ [1, 2, 3], [4, 5, 6] ]

@ Lic. Ricarde Thempsen

#### Acceso a los elementos

 Para acceder a los elementos de una lista se utiliza un subíndice:

a = numeros[0]

 El primer elemento de la lista siempre lleva el subíndice 0.

#### Acceso a los elementos

 Usar un subíndice negativo hace que la cuenta comience desde atrás:

 Usar subíndices fuera de rango provocará un error.

@ Lic. Ricarde Thempsen

#### Impresión de listas

- Las listas pueden imprimirse a través de ciclos while o for.
- Pero también pueden imprimirse directamente:

numeros = [1, 2, 3, 4] print(numeros) # [1, 2, 3, 4] print(\*numeros) # 1 2 3 4



Empaquetar consiste en asignar un conjunto de constantes o variables a una lista:

$$n1 = 5$$

$$n2 = 9$$

$$n3 = 4$$

numeros = [n1, n2, n3]

© Lie. Ricarde Thempsen

### Desempaquetado

Desempaquetar consiste en asignar los elementos de una lista a un conjunto de variables:

dias = ["lunes", "martes", "jueves"] d1, d2, d3 = dias



Las listas pueden concatenarse con el operador +:

$$lista1 = [1, 2, 3]$$

$$lista2 = [4, 5, 6]$$

@ Lic. Ricarde Thempsen

#### **Operaciones con listas**

La concatenación permite agregar nuevos elementos en una lista:

$$lista = [3, 4, 5]$$

El elemento debe encerrarse entre corchetes para que sea considerado como una lista.



 También pueden ser replicadas (repetidas) mediante el asterisco:

```
lista1 = [3, 4, 5]

lista1 = lista1 * 3

print(lista1) # [3, 4, 5, 3, 4, 5, 3, 4, 5]

lista2 = [0] * 5

print(lista2) # [0, 0, 0, 0, 0]
```

© Lie. Ricarde Thempsen

#### **Operaciones con listas**

Otra manera de agregar elementos consiste en usar el método *append*:



Por ser secuencias *mutables*, los elementos de una lista pueden ser modificados a través del subíndice:

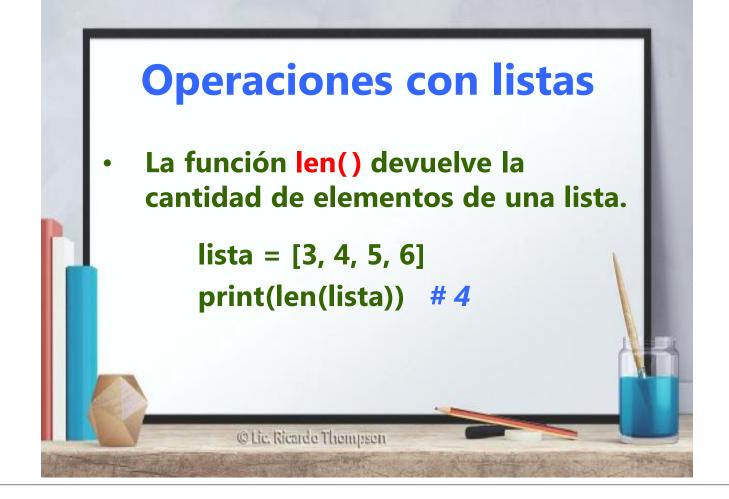
lista = [3, 4, 5] lista[1] = 7 print(lista) # [3, 7, 5]

@ Lic. Ricarde Thempsen

# Ejemplo 1

Leer un conjunto de números enteros, calcular su promedio e imprimir aquellos valores leídos que sean mayores que el promedio

```
lista = [ ]
suma = 0
cant = 0
n = int(input("Ingrese un número entero o -1 para terminar: "))
while n != -1:
  lista.append(n)
  suma = suma + n
  cant = cant + 1
  n = int(input("Ingrese un número entero o -1 para terminar: "))
if cant == 0:
  print("No se ingresaron valores")
else:
  prom = suma/cant
  print("Promedio:", prom)
  for i in range(cant):
    if lista[i] > prom:
       print(lista[i], end=" ")
  print( )
               © Lie. Ricarde Thempsen
```





La función sum() devuelve la suma de los elementos de la lista.

lista = [3, 4, 5, 6] print(sum(lista)) # 18

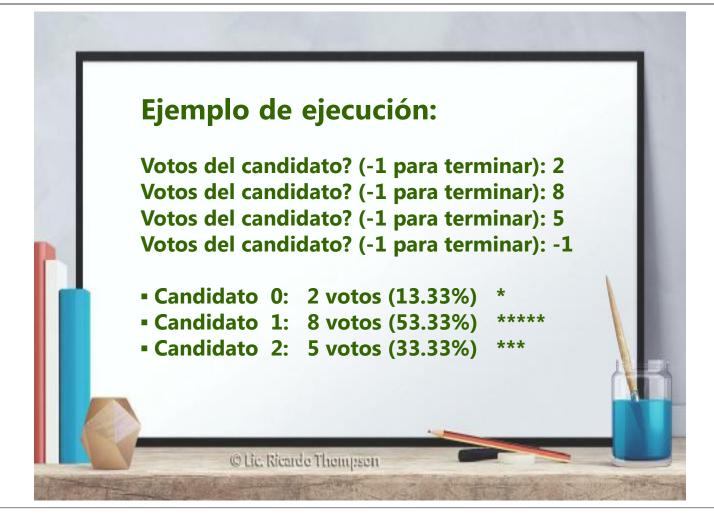
La lista debe contener números.

@ Lic. Ricarde Thempsen

# Ejemplo 2

Crear un gráfico de barras con los porcentajes obtenidos por cada candidato en una elección

```
votos = []
n = int(input("Votos del candidato? (-1 para terminar): "))
while n != -1:
  votos = votos + [n] # votos.append(n) sería similar
  n = int(input("Votos del candidato? (-1 para terminar) "))
print( )
# Calcular porcentajes e imprimir listado final
total = sum(votos)
for i in range(len(votos)):
  porcentaje = votos[i] * 100 / total
  print("- Candidato %d: %d votos (%.2f%%)"
         %(i, votos[i], porcentaje), end=" ")
  # Imprimir el gráfico de barras
  for j in range(int(porcentaje/10)):
    print("*", end="")
  print( )
             © Lie. Ricarde Thempsen
```



### **Operaciones con listas**

 La función max() devuelve el mayor de los elementos de la lista.

 La lista debe contener elementos homogéneos.

@ Lie. Ricarde Thempsen

#### **Operaciones con listas**

La función min() devuelve el menor de los elementos de la lista.

 La lista debe contener elementos homogéneos.



 Las funciones max() y min() también pueden ser utilizadas con un conjunto de valores, constantes o variables.

> mayor = max(4, 1, 7, 2) print(mayor) # 7

> > @ Lie. Ricarde Thempsen

#### **Operaciones con listas**

 La función list() convierte cualquier iterable en una lista.

> lista = list(range(5)) print(lista) # [0, 1, 2, 3, 4]

 Se puede usar con rangos, cadenas, tuplas, conjuntos, etc.



- El operador in permite verificar la presencia de un elemento.
- Devuelve True o False.

lista = [3, 4, 5, 6]
if 4 in lista:
 print("Hay un 4 en la lista")

@ Lie. Ricarde Thempsen

#### **Operaciones con listas**

- La ausencia de un elemento se comprueba con not in.
- Devuelve True o False.

lista = [3, 4, 5, 6]

if 7 not in lista:

print("No hay un 7 en la lista")

# Ejemplo 3

#### Uso del operador in

Escribir una función que reciba como parámetros dos números correspondientes al mes y año de una fecha y devuelva cuántos días tiene ese mes en ese año.

```
def obtenercantdias(mes, año):
    if mes in [1, 3, 5, 7, 8, 10, 12]: # Lista implícita
    dias = 31
    elif mes in [4, 6, 9, 11]:
        dias = 30
    elif mes==2:
        if (año%4==0 and año%100!=0) or (año%400==0):
            dias = 29
        else:
            dias = 28
    else:
        dias = -1 # Mes inválido
    return dias
```

## Métodos

- Un método es una función que pertenece a un objeto.
- Los métodos permiten manipular los datos almacenados en el objeto.
- Se escriben luego del nombre del objeto, separados por un punto.

@ Lie. Ricarde Thempsen

### Métodos

 El método append(<elem>) agrega un elemento al final de la lista.

> lista = [3, 4, 5] lista.append(6) print(lista) # [3, 4, 5, 6]



El método insert(<pos>, <elem>) inserta un elemento en la lista, en una posición determinada.

```
lista = [3, 4, 5]
lista.insert(2, 9)
print(lista) # [3, 4, 9, 5]
```

@ Lic. Ricarde Thempsen

### Métodos

• El método pop(<pos>) elimina y devuelve un elemento de la lista, identificado por su posición. Si la posición se omite se elimina el último elemento de la lista.

Da un error si la posición está fuera de rango.

## Métodos

 El método remove(<elem>) elimina la primera aparición de un elemento en la lista, identificado por su valor.

> lista = [3, 4, 5, 4] lista.remove(4) # queda [3, 5, 4]

Provoca un error si no existe.

@ Lic. Ricarde Thempsen

#### Métodos

 El método index(<elem>) busca un elemento y devuelve su posición.

> lista = [3, 4, 5] print(lista.index(5)) # 2

Provoca un error si no lo encuentra

### Métodos

 El método index() también permite elegir la región de búsqueda.

```
print(lista.index(5, 2)) # Inicio
print(lista.index(5, 2, 4)) # Inicio y fin
```

El valor final *no está incluido* en el intervalo de búsqueda.

© Lic. Ricarde Thempsen

#### Métodos

 El método count(<elem>) devuelve la cantidad de repeticiones de un elemento.

Devuelve 0 si no lo encuentra.



El método clear() elimina in situ\* todos los elementos de la lista.

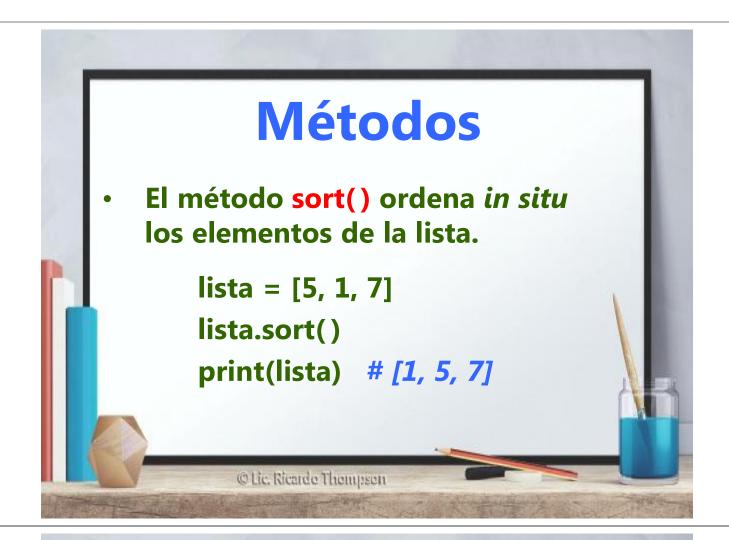
```
lista = [3, 4, 5]
lista.clear()
print(lista) # []
```

\* in situ: En su lugar, sin crear una lista nueva.

@ Lic. Ricarde Thempsen

### Métodos

el método reverse() invierte in situel el orden de los elementos de la lista.





#### Método sort

 El parámetro key= < clave > permite establecer el criterio de ordenamiento cuando éste no sea el valor del ítem.

```
lista = [1, 2, 3, 4, 5, 6]
lista.sort(key=lambda x: x%2)
print(lista) # [2, 4, 6, 1, 3, 5]
```

Nota: Python garantiza conservar el orden original ante claves iguales. A eso se lo denomina estabilidad del ordenamiento.

@ Lic. Ricarde Thempsen

#### Método sort

 Las funciones lambda son ideales para obtener claves de ordenamiento.
 También pueden usarse funciones normales.

```
nombres = ["Andrés", "Ariel", "Juan"]
nombres.sort(key=lambda x: x[-1])
print(nombres) # ["Ariel", "Juan", "Andrés"]
```

### Método sort

Si se utilizan funciones incorporadas no es necesario crear una función lambda.

```
nombres = ["Andrés", "Ariel", "Juan"]
nombres.sort(key=len)
print(nombres) # ["Juan", "Ariel", "Andrés"]
numeros = [3, -2, 4, -1]
numeros.sort(key=abs)
print(numeros) # [-1, -2, 3, 4]
```

© Lie. Ricarde Thempsen

#### Método sort

Cuando se necesite ordenar una lista por más de un atributo es necesario crear una lista con esos atributos, la que será utilizada como clave.

```
numeros = [9, 6, 7, 8, 1, 5, 4]
numeros.sort(key=lambda x:[x%2, x])
print(numeros) # [4, 6, 8, 1, 5, 7, 9]
```



Además del método sort de las listas, Python dispone de la función sorted(), que permite ordenar cualquier *iterable* (rango, lista, string, tupla, conjunto, diccionario, etc).

numeros = [9, 6, 7, 8, 1, 5, 4] ordenada = sorted(numeros) print(ordenada) # [1, 4, 5, 6, 7, 8, 9]

© Lie. Ricarde Thempsen

# Función sorted()

- La función sorted() también admite los parámetros reverse y key ya analizados con el método sort.
- En todos los casos sorted() devuelve una <u>lista</u> con los elementos ordenados.



- Son números generados, o inventados, por la computadora.
- Se utilizan cuando se requiere un factor de azar, por ejemplo en videojuegos, criptografía o simulación de eventos.

@ Lie. Ricarde Thempsen

#### Números al azar

- En Python hay varias funciones relacionadas con ellos.
- Todas pertenecen al módulo random:

import random

#### Números al azar

- random.random(): Devuelve un número real al azar dentro del intervalo [0, 1).
- random.randint(<min>, <max>):
   Devuelve un número entero al azar dentro del intervalo dado. El intervalo se considera cerrado.

@ Lie. Ricarde Thempsen

#### Números al azar

 random.choice(<secuencia>): Devuelve un elemento elegido al azar dentro de una secuencia pasada como parámetro.

opciones = ["Piedra", "Papel", "Tijera"]
situacion = random.choice(opciones)

La secuencia puede ser una lista, un string, una tupla, un rango, etc.



random.shuffle(<lista>): Mezcla in situ los elementos de una lista, es decir que altera la posición de los mismos.

lista = [3, 4, 5, 6] random.shuffle(lista) print(lista) # por ejemplo [6, 4, 5, 3]

@ Lic. Ricarde Thempsen

# Ejemplo 4

Uso de números al azar

Para un juego de generala se necesita desarrollar una función que simule el lanzamiento de los cinco dados de un cubilete. Escribir también un programa para verificar su funcionamiento.

```
import random

def lanzardados(cuantos):
    dados = []
    for i in range(cuantos):
        dados.append(random.randint(1, 6))
    return dados

# Programa principal
    jugada = lanzardados(5) # cinco dados
    print(jugada) # [5, 1, 2, 6, 3] o similar
```

