

Clase N° 4

Listas

Segunda Parte

© Lic. Ricardo Thompson

Rebanadas

- Una **rebanada** (list slicing) es una manera de referirse a un grupo de elementos pertenecientes a una lista.
- En lugar de usar un solo subíndice se utilizan dos o tres, separados por "dos puntos".

© Lic. Ricardo Thompson

Rebanadas

```
lista = [7, 8, 9, 10, 11, 12]
```

```
sublista = lista[2:5] # [9, 10, 11]
```

- Los dos subíndices indican el inicio y el fin de la rebanada.
- El subíndice final **no está incluido**.

© Lic. Ricardo Thompson

Rebanadas

- Cuando se usan tres subíndices, el tercero actúa como **incremento**.

```
lista = [7, 8, 9, 10, 11, 12, 13, 14]
```

```
sublista = lista[1: 6: 2]
```

```
print(sublista) # [8, 10, 12]
```

© Lic. Ricardo Thompson

Rebanadas

- Dejar en blanco alguno de los subíndices hace que se considere el extremo de la lista.

```
lista1 = [7, 8, 9, 10, 11, 12]
```

```
lista2 = lista1[3:] # [10, 11, 12]
```

```
lista3 = lista1[:3] # [7, 8, 9]
```

© Lic. Ricardo Thompson

Rebanadas

- Un incremento negativo toma los elementos *de atrás hacia adelante*.

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

```
invertida = original[::-1]
```

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

© Lic. Ricardo Thompson

Rebanadas

- Las rebanadas también pueden colocarse del lado izquierdo de la asignación, para reemplazar elementos.

```
lista = ['a', 'b', 'c', 'd', 'e']
```

```
lista[2:4] = ['C', 'D']
```

```
print(lista) # ['a', 'b', 'C', 'D', 'e']
```

© Lic. Ricardo Thompson

Rebanadas

- La cantidad de elementos reemplazados puede ser diferente de la original.

```
lista = ['a', 'b', 'c', 'd', 'e']
```

```
lista[2:4] = ['C', 'D', 'E']
```

```
print(lista) # ['a', 'b', 'C', 'D', 'E', 'e']
```

© Lic. Ricardo Thompson

Rebanadas

- Esta propiedad permite eliminar elementos consecutivos sin mucho esfuerzo.

```
lista = ['a', 'b', 'c', 'd', 'e']  
lista[2:4] = [ ]  
print(lista) # ['a', 'b', 'e']
```

© Lic. Ricardo Thompson

Rebanadas

- Las rebanadas también funcionan con variables.

```
lista = [1, 2, 3, 4, 5, 6, 7, 8]  
a = 3  
b = 5  
sublista = lista[a:b]  
print(sublista) # [4, 5]
```

© Lic. Ricardo Thompson

Rebanadas

- Una **rebanada nula** es una rebanada que no contiene *ningún elemento*.
- Se crean utilizando el mismo subíndice para inicio y fin.
- Recordar que el subíndice final no está incluido.

© Lic. Ricardo Thompson

Rebanadas

Ejemplo:

```
lista1 = ['a', 'b', 'c', 'd', 'e']
```

```
lista2 = lista1[2:2]
```

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

© Lic. Ricardo Thompson

Rebanadas

- Con esta técnica pueden insertarse varios elementos de una sola vez.

```
lista = ['a', 'b', 'c', 'd']
```

```
lista[2:2] = ['X', 'Y']
```

```
print(lista) # ['a', 'b', 'X', 'Y', 'c', 'd']
```

© Lic. Ricardo Thompson

Ejemplo 1

Uso de rebanadas

Obtener sublistas con los primeros y últimos N elementos de una lista.

© Lic. Ricardo Thompson

```
lista = list(range(10))
n = int(input("Cuántos elementos desea tomar? "))
comienzo = lista[:n] # N elementos del comienzo
final = lista[-n:] # N elementos del final
extremos = comienzo + final
print( )
print("Lista original:", lista)
print("Comienzo:", comienzo)
print("Final:", final)
print("Comienzo + Final:", extremos)
```

© Lic. Ricardo Thompson

Ejemplo de ejecución:

Cuántos elementos desea tomar? 3

Lista original: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Comienzo: [0, 1, 2]

Final: [7, 8, 9]

Comienzo + Final: [0, 1, 2, 7, 8, 9]

© Lic. Ricardo Thompson

Comparación de listas

- Las listas pueden ser comparadas como cualquier otra variable.
- La comparación se realiza **elemento a elemento**.

```
print([2, 3] > [1, 4])    # True  
print([2, 3] > [2, 4])    # False  
print([2, 4, 6] > [2, 4]) # True
```

© Lic. Ricardo Thompson

Copia de listas

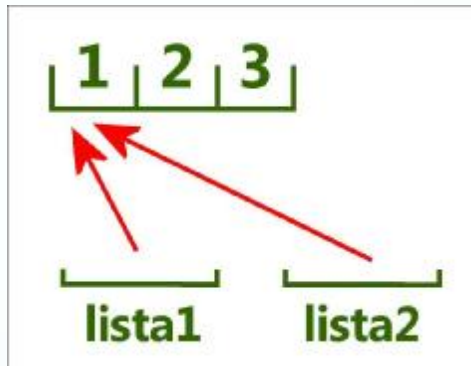
- Copiar una lista sólo copia la referencia al objeto.

```
lista1 = [1, 2, 3]  
lista2 = lista1  
lista2.append(4)  
print(lista1)  # [1, 2, 3, 4]
```

© Lic. Ricardo Thompson

Copia de listas

- Por lo tanto, las dos variables apuntan al mismo objeto en memoria.



© Lic. Ricardo Thompson

Copia de listas

- Esto puede verificarse mediante la función **id(<objeto>)**, que devuelve la *identidad* de un objeto y es equivalente a su dirección de memoria.

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

```
lista2 = lista1
```

```
print(id(lista1), id(lista2))
```

```
# por ejemplo 180464072 180464072
```

© Lic. Ricardo Thompson

Copia de listas

- Existen tres maneras para evitarlo.
- La primera consiste en realizar la copia a través de una *rebanada*.

```
lista1 = [1, 2, 3]
lista2 = lista1[ : ]
lista2.append(4)
print(lista1)  # [1, 2, 3]
```

© Lic. Ricardo Thompson

Copia de listas

- La segunda aprovecha la función *list()*.

```
lista1 = [1, 2, 3]
lista2 = list(lista1)
lista2.append(4)
print(lista1)  # [1, 2, 3]
```

© Lic. Ricardo Thompson

Copia de listas

- La tercera utiliza el método **copy()**.

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

```
lista2 = lista1.copy( )
```

```
lista2.append(4)
```

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

© Lic. Ricardo Thompson

Uso de for con listas

- La instrucción **for** puede utilizarse para recorrer listas sin necesidad de **range()**.
- En este caso la variable usada en el for recoge **todo el elemento** de la lista, y no su subíndice.
- Equivale al **foreach** de otros lenguajes.

© Lic. Ricardo Thompson

Uso de for con listas

Ejemplo:

```
vocales = ['a', 'e', 'i', 'o', 'u']
```

```
for letra in vocales:
```

```
    print(letra, end=" ") # a e i o u
```

© Lic. Ricardo Thompson

Uso de for con listas

- Puede usarse una rebanada para recorrer la lista parcialmente.

```
vocales = ['a', 'e', 'i', 'o', 'u']
```

```
for letra in vocales[1:4]:
```

```
    print(letra, end=" ") # e i o
```

© Lic. Ricardo Thompson

Uso de for con listas

- Si además del elemento se requiere su subíndice, puede usarse la función ***enumerate()***.

```
lista = ['a', 'e', 'i', 'o', 'u']
```

```
for i, letra in enumerate(lista):
```

```
    print(i, letra)    # 0 a, 1 e, 2 i...
```

© Lic. Ricardo Thompson

Uso de for con listas

- ***enumerate()*** devuelve una pareja de valores formada por el subíndice y el elemento correspondiente.

(<subíndice>, <elemento>)

- Esta pareja de valores recibe el nombre de *tupla*, y se *desempaqueta* en dos variables.

© Lic. Ricardo Thompson

Instrucción del

- La instrucción **del** permite borrar elementos o listas completas.

lista = [7, 8, 9, 10, 11, 12]

del lista[1] *# borra el 8*

del lista *# borra toda la lista*

- del también soporta rebanadas.

© Lic. Ricardo Thompson

Instrucción pass

- La instrucción **pass** no hace nada.
- Puede usarse en ciclos vacíos o para representar código aún no escrito.

def calcularsalario(empleado):

pass *# aún no implementado*

© Lic. Ricardo Thompson

Función map

- La función **map** aplica una función cualquiera a todos los elementos de una lista.
- Su sintaxis es la siguiente:

```
<lista2> = list(map(<función>, <lista1>))
```

© Lic. Ricardo Thompson

Función map

- Ejemplo:
 `numeros = [1, 2, 3, 4]`
 `raices = list(map(lambda x: x**(1/2), numeros))`
- Esto equivale a:
 `numeros = [1, 2, 3, 4]`
 `raices = []`
 for i in numeros:
 `raices.append(i**(1/2))`

© Lic. Ricardo Thompson

Función map

- Aunque pueden utilizarse funciones normales, las funciones lambda son ideales para estos casos.
- La función `list()` es necesaria para convertir a formato de lista el objeto devuelto por `map()`.

© Lic. Ricardo Thompson

Función filter

- La función ***filter*** selecciona algunos elementos de una lista para crear una nueva lista con ellos.
- Los elementos de la lista original que se añaden a la nueva lista son aquellos que devuelven *True* al aplicarles una función.

© Lic. Ricardo Thompson

Función filter

Ejemplo:

```
numeros = [0, 1, 2, 3, 4, 5]
r_ent = list(filter(lambda x: x**(1/2)==int(x**(1/2)), numeros))
print(r_ent)  # [0, 1, 4]
```

Nota: r_ent significa raíces enteras

© Lic. Ricardo Thompson

Listas por comprensión

- Las **listas por comprensión** son una manera matemática para crear listas, adoptada de los conjuntos.

```
cuadrados = [i**2 for i in range(6)]
print(cuadrados)  # [0, 1, 4, 9, 16, 25]
```

© Lic. Ricardo Thompson

Listas por comprensión

- La sintaxis para construirlas es:

<lista> = [<expr> for <var> in <secuencia>]

- La expresión **<expr>** representa alguna operación matemática que se aplica a cada elemento **<var>** de **<secuencia>**. El resultado de esta expresión se agregará a **<lista>**.
- Los corchetes son necesarios porque se está creando una lista.

© Lic. Ricardo Thompson

Listas por comprensión

- Puede agregarse un **if** para filtrar elementos.

```
cubospares = [i**3 for i in range(11) if i**3 % 2 == 0]  
print(cubospares) # [0, 8, 64, 216, 512, 1000]
```

© Lic. Ricardo Thompson

Listas por comprensión

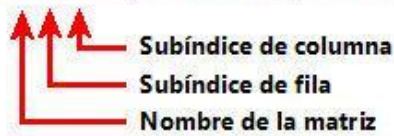
- En muchos casos las funciones *map* y *filter* tratadas anteriormente pueden ser reemplazadas por listas por comprensión.

© Lic. Ricardo Thompson

Matrices

- Una **matriz** es una estructura de datos formada por filas y columnas.

	Columna 0	Columna 1	Columna 2	Columna 3
Fila 0	A[0][0]	A[0][1]	A[0][2]	A[0][3]
Fila 1	A[1][0]	A[1][1]	A[1][2]	A[1][3]
Fila 2	A[2][0]	A[2][1]	A[2][2]	A[2][3]



© Lic. Ricardo Thompson

Matrices

- A diferencia de la mayoría de los lenguajes de programación, Python no tiene soporte para matrices.
- Por eso se las simula contruyendo una *lista de listas*.

© Lic. Ricardo Thompson

Matrices

- Una lista de listas es una lista donde sus elementos son, a su vez, listas.
- Se necesitan dos subíndices, el primero se refiere a las filas y el segundo a las columnas. Ambos comienzan desde 0.

© Lic. Ricardo Thompson

Matrices

- En este curso todas las matrices serán regulares, es decir que todas las filas tendrán la misma cantidad de columnas.
- La creación de la misma se puede hacer en forma *estática* o *dinámica*.

© Lic. Ricardo Thompson

Creación de matrices

**# Alternativa 1: Crear la matriz como
una lista de listas en forma estática**

```
matriz = [ [0,0,0,0],  
            [0,0,0,0],  
            [0,0,0,0] ]
```

----- Fin creación de la matriz -----

© Lic. Ricardo Thompson

Creación de matrices

*# Alternativa 2: Crear la matriz como una
lista de listas en forma dinámica*

```
filas = 3
columnas = 4
matriz = [ ]
for f in range(filas):
    matriz.append([ ])
    for c in range(columnas):
        matriz[f].append(0)
# ----- Fin creación de la matriz -----
```

© Lic. Ricardo Thompson

Creación de matrices

*# Alternativa 3: Similar a la 2 pero usando
el poder de replicación de Python*

```
filas = 3
columnas = 4
matriz = [ ]
for f in range(filas):
    matriz.append( [0] * columnas )
# ----- Fin creación de la matriz -----
```

© Lic. Ricardo Thompson

Creación de matrices

Alternativa 4: Usando listas por comprensión

filas = 3

columnas = 4

matriz = [[0] * columnas for i in range(filas)]

----- Fin creación de la matriz -----

© Lic. Ricardo Thompson

Operaciones con matrices

- **Una vez creada la matriz la rellenaremos con números ingresados a través del teclado. Luego la imprimiremos por pantalla.**
- **Ambas tareas serán realizadas a través de funciones.**

© Lic. Ricardo Thompson

Operaciones con matrices

```
# Este código va dentro del programa  
# principal, después de haber creado  
# la matriz con cualquiera de las  
# cuatro alternativas analizadas
```

```
rellenarmatriz(matriz)  
imprimirmatriz(matriz)
```

© Lic. Ricardo Thompson

Lectura de datos

```
def rellenarmatriz(matriz):
```

```
# Autodetectamos el tamaño de la matriz
```

```
filas = len(matriz)
```

```
columnas = len(matriz[0])
```

```
for f in range(filas):
```

```
    for c in range(columnas):
```

```
        n = int(input("Ingrese un número: "))
```

```
        matriz[f][c] = n
```

© Lic. Ricardo Thompson

Impresión de la matriz

```
def imprimirmatriz(matriz):  
    # Autodetectamos el tamaño de la matriz  
    filas = len(matriz)  
    columnas = len(matriz[0])  
    for f in range(filas):  
        for c in range(columnas):  
            print("%3d" %matriz[f][c], end="")  
        print( )
```

© Lic. Ricardo Thompson

Impresión de la matriz

*# Otra manera de imprimir la matriz,
sin utilizar subíndices (Sólo para lectura).*

```
def imprimirmatriz(mat):  
    for fila in mat:  
        for elem in fila:  
            print("%3d" %elem, end="")  
        print( )  
    print( )
```

© Lic. Ricardo Thompson

Programa completo

```
def rellenarmatriz(matriz):  
    # Autodetectamos el tamaño de la matriz  
    filas = len(matriz)  
    columnas = len(matriz[0])  
    for f in range(filas):  
        for c in range(columnas):  
            n = int(input("Ingrese un número: "))  
            matriz[f][c] = n  
  
def imprimirmatriz(matriz):  
    # Autodetectamos el tamaño de la matriz  
    filas = len(matriz)  
    columnas = len(matriz[0])  
    for f in range(filas):  
        for c in range(columnas):  
            print("%3d" %matriz[f][c], end="")  
        print( )  
  
# Programa principal  
filas = 3  
columnas = 4  
matriz = [ ]  
for f in range(filas):  
    matriz.append( [0] * columnas )  
# --- Fin creación de la matriz -----  
rellenarmatriz(matriz)  
imprimirmatriz(matriz)
```

© Lic. Ricardo Thompson

Ejercitación

- **Práctica 2: Ejercicios 10 a 15**
- **Práctica 3: Completa**

© Lic. Ricardo Thompson