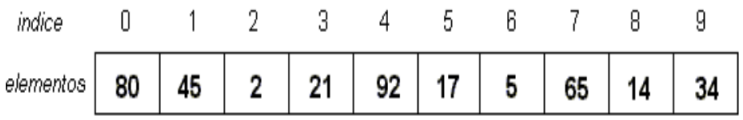
Clase 16: Estructuras Indexadas

(Cap. 12 del Apunte)

Estructuras Indexadas

Listas de python (arreglos)



Las listas de Python son estructuras definidas nativamente en el lenguaje Python que

- permiten acceder a los valores almacenados en la lista a través de un índice
    - Ej: strings
- las instrucciones `'for'` y `'while'` nos permiten recorrer ('**iterar**') sobre los valores de una lista
- los índices son números enteros correlativos y parten en cero

Ej: Crear una lista vacía:

```
unaLista = list() # lista vacia


otraLista = [] # lista vacia
print(otraLista)
```

También se pueden crear listas con valores ya insertados, como se muestra en el ejemplo siguiente. Recuerde que las listas de Python permiten insertar datos de cualquier tipo en sus casilleros:

```
Enteros = [1, 2, 3, 4, 5] # lista de int


Strings = ['casa', 'arbol', 'planta', 'auto'] # lista de str


Todo = ['a', 17, True, 9.5] # lista con datos de distinto tipo
```

Indices (comienzan en 0)

Para acceder a los valores en una lista se utiliza el índice del casillero correspondiente. No olvide que el **primer valor** de la lista **corresponde al índice cero**. Por ejemplo:

```
lista = ['ana', 'maria', 'luisa', 'teresa']
lista[0] # primer valor es [0] (matlab es [1])
```

```
lista[3]
```

```
lista[4]
```

Note en el código anterior que cuando se intentó acceder a un casillero de la lista con un valor de índice inválido (por ejemplo, se sale del rango de los índices de la lista), se produjo un **`IndexError`**.

Las **listas** son estructuras **mutables**, por lo que se pueden modificar los valores almacenados en sus casilleros:

```
lista = [10, 20, 30, 40, 50, 60]
lista[3] = 100
print(lista)
```

Concatenación

Dos listas se pueden **concatenar con el operador '+'**, o se puede **repetir varias veces el contenido de una lista multiplicándola por un escalar**:

```
lista1 = [10, 20]
lista2 = [50, 60]
lista3 = lista2 + lista1  # concatenacion
print(lista3)
```

```
lista1 * 3 # concatena lista consigo misma
```

```
lista1 + [30]
```

Comparación

True porque 20>2

```
L=[20,30,10]
L>[2,4,6] #compara elemento a elemento
```

```
L2 = [30,20,5]
L>L2
```

```
"ARTE" > "ROMA"
```

Sublista (slice)

Podemos acceder a un *trozo* o rango de la lista. Usando la notación **`[i:j]`**

```
print(L)
print(L[1:3]) #lista c/valores entre indices 1 y 2 (3-1)
```

```
L1 = [10,20,30,40,50,60]
print(L1[2:5])
```

Otro ejemplo:

```
cuadrados = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
cuadrados[3:6] # solo mostrar indices 3, 4 y 5
```

Para conocer el largo de una lista (esto es, cuántos valores contiene), se utiliza la función **`len`** de Python:

```
lista = ['ana', 'maria', 'luisa', 'teresa']
len(lista)
```

```
lista[len(lista)-1]
```

Aliasing

```
L=[20,30,10]
```

```
M=L #M es un alias de L
M
```

```
M[0]=200
M
```

```
L
```

Una lista puede contener otras listas

```
M = [[1,2,3],10,20]
print(M[0])
print(M[1])
```

```
matriz = [[10, 20, 30], [40, 50, 60], [70, 80, 90]]
matriz[1][2]
```

Funciones predefinidas

Ejemplos:

```
L = [8,9,7,9]
total = sum(L) # Suma los elementos
print(total)
```

```
L.index(9) # retorna primer indice de 9
```

```
L.append(6) # agrega un 6 como elemento al final
L
```

```
L.extend([10]) # extiende la lista L con la lista [10]
L
```

```
L2 = [8,9,7,9,6]
L2 = L2 + [10]
print(L2)
```

```
B1 = [1,2,3]
B2 = [4,5]
B1.extend(B2)
print(B1)
```

```
A1 = [1,2,3]
A2 = [4,5]
A1.append(A2)
print(A1)
```

```
L=[8,9,7,9,6,10]
```

```
e = L.pop(1) # saca (extrae) el valor en la posición 1, modifica L
print(e)
```

```
L
```

```
L.remove(9)  # extrae de la lista L el primer valor 9 encontrado
L
```

```
L.reverse()
L
```

```
L.sort()
L
```

```
help(list)
```

Función predefinida: range (rango)

Una función útil predefinida de Python es la función **`range`**, que permite crear una lista de Python **`list`** que contiene una progresión aritmética de enteros.

```
list(range(10))
```

Esta función provee tres contratos distintos:

```
range(stop): int -> iterador(int)
range(start, stop): int int -> iterador(int)
range(start, stop, step): int int int -> iterador(int)
```

Un `iterador` es un tipo especial de lista para usar en situaciones especiales (como en un for, el cual veremos más abajo).

IMPORTANTE: Por simplicidad *y por ahora* podemos suponer que range retorna una lista. Por qué? Siempre podemos transformar un iterador en una lista usando `list(iterador)`.

La primera versión

```
range(stop): int -> iterador(int)
```

permite crear una lista que comiencia en cero y termina en **`stop - 1`**.

```
list(range(20))
```

Note que para una lista de largo n, **`list(range(n))`** retorna la lista **`[0, 1, 2, ..., n-1]`**, que corresponden a los índices válidos para acceder a los casilleros de la lista.

La segunda versión

```
 range(start, stop): int int -> iterador(int)
```

permite crear una lista **`[start, start+1, ..., stop-2, stop-1]`**.

```
list(range(1,10))
```

En la tercera versión

Forma general: `range(inicial,final,incremento)`

```
 range(start, stop, step): int int int -> iterador(int)
```

el parámetro **`step`** específica el incremento (o decremento) entre números consecutivos de la lista asociada.

```
list(range(0,10,2))
```

si se omiten inicial e incremento?

Se considera `inicial=0,incremento=1`.

```
list(range(3))
```

```
list(range(0,3,1))
```

¿incremento<0?

```
list(range(3,0,-1))
```

Instrucción 'for'

La instrucción **`for`** de Python permite iterar sobre una estructura indexada.

Sintaxis:

```
for variable in lista:
  # bloque de instrucciones
```

donde **`variable`** es una variable y **`lista`**

Semántica:

La variable toma consecutivamente todos los valores de la lista. Para cada valor ejecuta las instrucciones

es el identificador de la estructura sobre la cual se va a iterar. En cada iteración del ciclo, se le asignará a la variable un valor almacenado en la la estructura indexada (por ejemplo, los valores almacenados en una lista), que se indica después de la instrucción **`in`**. Por ejemplo:

```
lista = [10, 20, 30, 40, 50, 60, 70]
for valor in lista:
        print("primer instruccion del bloque")
        print(valor)
        print("ultima instruccion del bloque")
```

En el ejemplo anterior, la instrucción del ciclo se ejecuta siete veces, una vez por cada asignación realizada a la variable **`valor`**.

- En la primera iteración, la variable valor vale 10.
- En la segunda iteración, valor valor 20.
- etc.
- En la séptima iteración, valor vale 70.

Instrucción for usando range( ):

```
for i in range(n):
  instrucciones
```

en cada iteración, **`i`** toma un valor de la lista **`[0,1,…,n-1]`**

Note que es posible obtener el mismo resultado anterior si es que a la variable se le asigna un valor de una lista "retornada" por **`range`**, y luego dicho valor se ocupa como indice para acceder a los distintos casilleros de la lista:

```
L = [10, 20, 30, 40, 50]
for indice in range(len(L)):
        print("El valor en indice=",indice,"es",L[indice])
```

Ejemplo: usar range pero con incrementos negativos.

```
L = [10, 20, 30, 40, 50, 60]
#escribir valores de L en orden inverso
for i in range(len(L)-1, -1, -1):
    print(L[i])
```

Trucos de Python: Permite crear listas con "for" y "range" con sintaxis especial.

Ejemplo: lista con todos los primeros 11 cuadrados perfectos, comenzando por 0.

```
cuadrados1 = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]  # muy latero
print(cuadrados1)
```

```
cuadrados2 = [x**2 for x in range(0,11)]
print(cuadrados2)
```

```
# cómo acordarse? Se parece a
for x in range(0,11):
  print(x**2)
```