Listas

Jonatan Goméz Perdomo, Ph.D.
jgomezpe@unal.edu.co

Arles Rodríguez, Ph.D.
aerodriguezp@unal.edu.co

Camilo Cubides, Ph.D.(c)
eccubidesg@unal.edu.co

Grupo de investigación en vida artificial – Research Group on Artificial Life – (Alife)
Departamento de Ingeniería de Sistemas e Industrial
Facultad de Ingeniería
Universidad Nacional de Colombia

Agenda

Definición

Una lista es una secuencia de elementos que puede almacenar datos heterogéneos tales como: enteros, reales, cadenas, tuplas, diccionarios y otros más, inclusive otras listas. Una lista se escribe como la secuencia de datos a mantener, separados por una coma (,), y delimitada por los paréntesis cuadrados (corchetes). A diferencia de las cadenas y las tuplas, las listas son mutables, esto es, se pueden modificar después de definidas.

`[]` : La lista vacía.

`["Este es un texto"]` : Una lista con un elemento. Python no requiere la coma final para considerarla una lista, así como si ocurre en las tuplas.

`['Una cadena', 123]` : Una lista de dos elementos, el primero una cadena y el segundo un número entero.

`[1, 2, 3, 4.5, 'hola', 'a']` : Una lista de seis elementos.

Variables

Una lista se puede asignar a una variable.

`x = []` : Le asigna la lista vacía a la variable `x`.

`lista = [1, 2, 3, 4.5, 'hola', 'a']` : Le asigna la lista de seis elementos a la variable `lista`.

`a = [1, 2, 3]` : Le asigna la lista `[1,2,3]` a la variable `a`.

Listas anidadas

Es posible crear Listas que tengan Listas como elementos (anidadas). Para el programa

```
lista1 = [0, 1, 2, 3]
lista2 = ['A', 'B', 'C']
lista3 = [lista1, lista2]
print(lista3)
print(lista3[0])
print(lista3[1])
print(lista3[1][0])
```

La salida obtenida es

```
[[0, 1, 2, 3], ['A', 'B', 'C']]
[0, 1, 2, 3]
['A', 'B', 'C']
A
```

Agenda

Concatenar +

Concatena dos listas produciendo una nueva lista. Para el programa

```
list1 = ['A', 'B', 'C', 'E']
list2 = [1,2,3,4,5]
list3 = list1 + list2
print(list3)
```

La salida obtenida es

```
['A', 'B', 'C', 'E', 1, 2, 3, 4, 5]
```

Agregar al final (extend)

El método extend agrega una lista al final de otra lista, la operación afecta la lista invocante. Para el programa

```
nombres = ['Antonio', 'María', 'Mabel']
otros_nombres = ['Barry', 'John', 'Guttag']
nombres.extend(otros_nombres)
print(nombres)
print(otros_nombres)
```

La salida obtenida es

```
['Antonio', 'María', 'Mabel', 'Barry', 'John', 'Guttag']
['Barry', 'John', 'Guttag']
```

Repetir *

Crea una lista con múltiples copias de una lista, tantas como se defina.
Para el programa

```
list1 = [1, 2, 3, 4, 5]
list2 = list1 * 3
print(list2)
list3 = ['Abc', 'Bcd']
list4 = list3 * 2
print(list4)
```

La salida obtenida es

```
[1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5]
['Abc', 'Bcd', 'Abc', 'Bcd']
```

Comparar I

Se usan los operadores convencionales (`<`, `<=`, `>`, `>=`, `==`, `!=`) para comparar listas usando el orden **lexicográfico**. En el orden lexicográfico, se comparan de izquierda a derecha uno a uno los elementos de la lista, mientras sean iguales. En el caso que no sean iguales, si el elemento de la primera lista es menor que el de la segunda, la primer lista se considera la menor, si el elemento es mayor la primer lista se considera la mayor. Si todos los elementos son iguales, se considera que las listas son iguales.

Comparar II

Para el programa

```
print(['Rojas', 123] < ['Rosas', 123])
print(['Rosas', 123] == ['rosas', 123])
print(['Rosas', 123] > ['Rosas', 23])
print(['Rosas', "123"] > ['Rosas', '23'])
```

La salida obtenida es

```
True
False
True
False
```

Comparar (`is`)

Se puede usar el operador `is` para determinar si dos listas referencian al mismo objeto. Para el programa de la izquierda, la salida obtenida se muestra a la derecha

```
a = ['Rojas', 123]
b = ["Rojas", 123]
c = a
print(a == b)
print(a is b)
print(a == c)
print(a is c)
print(b == c)
print(b is c)
```

```
True
False
True
True
True
False
```

Subíndice [ ]

Accede los elementos de una lista. Si la posición que se envía es negativa, lo considera desde el final. Para el programa

```
avengers = ["Ironman", "Thor", "Ant-man", "Hulk"]
print(avengers[0])
print(avengers[3])
print(avengers[-1])
print(avengers[-3])
```

La salida obtenida es

```
Ironman
Hulk
Hulk
Thor
```

Agenda

Consultando una lista

Es posible determinar si un elemento se encuentra en una lista. Para el programa

```
text = ['cien', 'años', 'de', 'soledad']
if 'años' in text:
   print('Si está en la lista')
else:
   print('No está en la lista')
```

La salida obtenida es

```
Si está en la lista
```

Consultando una lista

Es posible determinar si un elemento no se encuentra en una lista. Para el programa

```
text = ['cien', 'años', 'de', 'soledad']
if 'sien' not in text:
    print('No está en la lista')
else:
    print('Está en la lista')
```

La salida obtenida es

```
No está en la lista
```

Iterando una lista

Es posible iterar una lista usando el ciclo `for`. Para el programa

```
s = ["hola", "amigos", "mios"]
for palabra in s:     # para cada palabra de la lista
   print(palabra, end = ', ')
```

La salida obtenida es

```
hola, amigos, mios,
```

Creando una lista con un ciclo para (`for`)

Es posible asignarle una lista a una variable, usando la asignación, el ciclo `for` y el concepto de lista. Para el programa

```
d = 10
desplaza = [d + x for x in range(5)]
print(desplaza)
potencias = [3 ** x for x in range(2, 6)]
print(potencias)
```

La salida obtenida es

```
[10, 11, 12, 13, 14]
[9, 27, 81, 243]
```

Asignando múltiples variables desde una lista

Es posible asignarle los valores a un grupo de variables usando la asignación y el concepto de lista (desempacando). Para el programa

```
lista = [1, -2, 3]
a, b, c = lista
print('a =', a)
print('b =', b)
print('c =', c)
```

La salida obtenida es

```
a = 1
b = -2
c = 3
```

Asignando múltiples variables desde una lista

Es posible asignar los valores de un grupo de variables usando la asignación, el ciclo `for` y el concepto de lista. Para el programa

```
lista = [11, 9, -2, 3, 8, 5]
var1, var2, var3 = [lista[i] for i in (1, 3, 5)]
print("var1 =", var1, ", var2 =", var2, ", var3 =", var3)
var1, var2, var3 = [lista[i] for i in range(0, 6, 2)]
print("var1 =", var1, ", var2 =", var2, ", var3 =", var3)
```

La salida obtenida es

```
var1 = 9 , var2 = 3 , var3 = 5
var1 = 11 , var2 = -2 , var3 = 8
```

Listas y funciones

Es posible retornar más de un valor en una función usando el concepto de lista. Para el programa

```
def minmax(a, b):
    if a < b:
        return [a, b]
    else:
        return [b, a]
x, y = minmax(5, 13)
print('min =', x, ",", 'max =', y)
x, y = minmax(12, -4)
print('min =', x, ",", 'max =', y)
```

La salida obtenida es

```
min = 5 , max = 13
min = -4 , max = 12
```

Agenda

Longitud (`len`)

La función `len` determina la dimensión (longitud) de una lista. Para el programa

```
lista = [1, 2, 3, 4]
nombre = ["Minch", "Yoda"]
trabajo = ["Stars", "War", "Movie"]
empty = []
print(len(lista))
print(len(nombre))
print(len(trabajo))
print(len(empty))
```

La salida obtenida es

```
4
2
3
0
```

Cambiando elementos

Es posible cambiar un elemento en una posición de una lista. Para el programa

```
lista = ['E', 'l', 'm', 'e', 'j', 'o', 'r']
lista[0] = 'e'
print(lista)
lista[4] = 'l'
print(lista)
lista[-1] = 's'
print(lista)
```

La salida obtenida es

```
['e', 'l', 'm', 'e', 'j', 'o', 'r']
['e', 'l', 'm', 'e', 'l', 'o', 'r']
['e', 'l', 'm', 'e', 'l', 'o', 's']
```

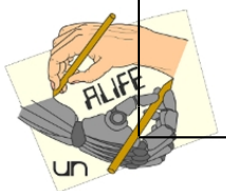
Agregando elementos (append)

El método append permite agregar elementos al final de una lista. Para el programa

```
nombres = ['Antonio', 'Johan']
nombres.append('Monica')
print(nombres)
nombres.append('Maria')
print(nombres)
nombres.append('Mabel')
print(nombres)
```

La salida obtenida es

```
['Antonio', 'Johan', 'Monica']
['Antonio', 'Johan', 'Monica', 'Maria']
['Antonio', 'Johan', 'Monica', 'Maria', 'Mabel']
```



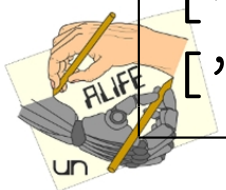
Insertando elementos (insert)

El método insert permite insertar (agregar) elementos en una posición específica de una lista. Para el programa

```
nombres = ['Antonio', 'Johan', 'Maria']
nombres.insert(0, 'Guttag')
print(nombres)
nombres.insert(2, 'Peter')
print(nombres)
nombres.insert(len(nombres)//2, 10)
print(nombres)
```

La salida obtenida es

```
['Guttag', 'Antonio', 'Johan', 'Maria']
['Guttag', 'Antonio', 'Peter', 'Johan', 'Maria']
['Guttag', 'Antonio', 10, 'Peter', 'Johan', 'Maria']
```



Eliminando elementos (remove)

El método remove permite eliminar la primera aparición (de izquierda a derecha) de un elemento de una lista. Para el programa

```
lista = ['a', 'e', 'i', 'o', 'u', 'i', 'x']
lista.remove('x')
print(lista)
lista.remove('i')
print(lista)
lista.remove('i')
print(lista)
```

La salida obtenida es

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

Sublistas (slice)

La función `slice` obtiene una porción (sublista) de una lista. La definición es igual a la función `slice` de tuplas `[inicio:fin:incremento]`. Para el programa

```
avengers = ["Ironman", "Thor", "Ant-man", "Hulk"]
print(avengers[:2])
print(avengers[1:3])
print(avengers[3:3])
print(avengers[::-1])
```

La salida obtenida es

```
['Ironman', 'Thor']
['Thor', 'Ant-man']
[]
['Hulk', 'Ant-man', 'Thor', 'Ironman']
```

Contando (count)

El método count obtiene las veces que un elemento se encuentra en una lista. Para el programa

```
lista = [4, 3, 8, 8, 2, 5, 4, 6, 8, 9]
print(lista.count(2))
print(lista.count(8))
print(lista.count(5))
print(lista.count(7))
```

```
1
3
1
0
```

Buscando (index)

El método index obtiene la primera ocurrencia de un elemento en una lista. Para el programa

```
lista = [4, 3, 8, 8, 2, 5, 4, 6, 8, 9]
print(lista.index(2))
print(lista.index(8))
print(lista.index(5))
```

La salida obtenida es

```
4
2
5
```

En caso de que el objeto que se esté buscando no se encuentre en la lista, se generará una excepción.

Máximo y mínimo (`max, min`)

El método `max/min` obtiene el máximo/mínimo elemento de una lista.
Para el programa

```
t = [4, 5, -1, 6, 7]
print(max(t))
print(min(t))
```

La salida obtenida es

```
7
-1
```

Ordenando (sort)

El método sort ordena una lista. Se le puede indicar si ascendente o descendentemente, al igual que el criterio (llave) usado para ordenar. Aquí solo se considera el ordenamiento sencillo. Para el programa

```
lista = [4, 5, -1, 6, 7]
lista.sort()                    # De menor a mayor
print(lista)
lista.sort(reverse = True) # De mayor a menor
print(lista)
```

La salida obtenida es

```
[-1, 4, 5, 6, 7]
[7, 6, 5, 4, -1]
```

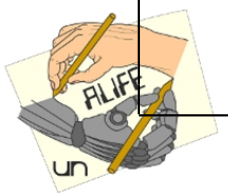
Convertir a lista (`list`)

El método `list` se usa para crear listas a partir de otros objetos, aquí se usa para convertir una cadena de caracteres y una tupla a listas. Para el programa

```
magician = 'Dumbledore'
lm = list(magician)
print(lm)
t = (1, 2, 3, 4)
lt = list(t)
print(lt)
```

La salida obtenida es

```
['D', 'u', 'm', 'b', 'l', 'e', 'd', 'o', 'r', 'e']
[1, 2, 3, 4]
```



Remover en una posición (pop)

En el caso en el que se requiera eliminar un elemento de una posición específica de la lista. El método pop recibe como parámetro la posición del elemento a remover. Las posiciones en las listas empiezan en cero. Si no se especifican parámetros, se remueve el último elemento de la lista.

```
nombres = ['Antonio','Johan','Monica','María','Mabel']
nombres.pop(1)                                  #remueve a Johan
print(nombres)
nombre_borrado = nombres.pop()                  # remueve a Mabel
print(nombre_borrado + " ha sido eliminada de la lista.")
print(nombres)
```

La salida obtenida es

```
['Antonio', 'Monica', 'María', 'Mabel']
Mabel ha sido eliminada de la lista.
['Antonio', 'Monica', 'María']
```

Sugerencia

Se sugiere consultar un manual de Python o de sus librerías para determinar si ya existe un método para lo que se quiera realizar con una lista.