

 [python-unsam](#) / **UNSAM_2020c2_Python****Code**

Issues

Pull requests 1


Actions

Projects

Wiki

Security

Insights

 master ▾

...

[UNSAM_2020c2_Python](#) / [Notas](#) / [01_Introduccion](#) / **05_Listas.md****rgrimson** actualizo repo 1 contributor

Raw

Blame



419 lines (312 sloc) 11.4 KB

[Contenidos](#) | [Anterior \(4 Cadenas\)](#) | [Próximo \(6 Cierre de la primer clase\)](#)

1.5 Listas

En esta sección estudiaremos listas que es el tipo de datos primitivo de Python para guardar colecciones ordenadas de valores.

Creación de Listas

Usá corchetes para definir una lista:

```
nombres = [ 'Rosita', 'Manuel', 'Luciana' ]  
nums = [ 39, 38, 42, 65, 111]
```

A veces las listas son creadas con otros métodos. Por ejemplo, los elementos de una cadena pueden ser separados en una lista usando el método `split()` :

```
>>> line = 'Pera,100,490.10'  
>>> row = line.split(',') #la coma indica el elemento que separa  
>>> row  
['Pera', '100', '490.10']  
>>>
```

Operaciones con listas

Las listas pueden almacenar elementos de cualquier tipo. Podés agregar nuevos elementos usando `append()` :

```
nombres.append('Mauro')      # Lo agrega al final
```

Usá el símbolo de adición `+` para concatenar listas:

```
s = [1, 2, 3]
t = ['a', 'b']
s + t           # [1, 2, 3, 'a', 'b']
```

Las listas se indexan con número enteros, comenzando en 0.

```
nombres = [ 'Rosita', 'Manuel', 'Luciana' ]

nombres[0] # 'Rosita'
nombres[1] # 'Manuel'
nombres[2] # 'Luciana'
```

Los índices negativos cuentan desde el final.

```
nombres[-1] # 'Luciana'
```

Podés cambiar cualquier elemento de una lista.

```
nombres[1] = 'Juan Manuel'
nombres           # [ 'Rosita', 'Juan Manuel', 'Luciana' ]
```

Y podés insertar elementos en una posición. Acordate que los índices comienzan a contar desde el 0.

```
nombres.insert(2, 'Iratxe') # Lo inserta en la posición 2.
nombres.insert(0, 'Iratxe') # Lo inserta como primer elemento.
```

La función `len` permite obtener la longitud de una lista.

```
nombres = ['Rosita', 'Manuel', 'Luciana']
len(nombres) # 3
```

Test de pertenencia a la lista (`in` , `not in`).

```
'Rosita' in nombres      # True
'Diego' not in nombres   # True
```

Se puede replicar una lista (`s * n`).

```
s = [1, 2, 3]
s * 3  # [1, 2, 3, 1, 2, 3, 1, 2, 3]
```

Iteradores de listas y búsqueda

Usá el comando `for` para iterar sobre los elementos de una lista.

```
for nombre in nombres:
    # usá nombre
    # e.g. print(nombre)
    ...
```

Para encontrar rápidamente la posición de un elemento en una lista, usá `index()` .

```
nombres = ['Rosita', 'Manuel', 'Luciana']
nombres.index('Luciana')  # 2
```

Si el elemento está presente en más de una posición, `index()` te va a devolver el índice de la primera aparición. Si el elemento no está en la lista se va a generar una excepción de tipo `ValueError` .

Borrar elementos

Podés borrar elementos de una lista tanto usando el valor del elemento como su posición:

```
# Usando el valor
nombres.remove('Luciana')

# Usando la posición
del nombres[1]
```

Al borrar un elemento no se genera un hueco. Los siguientes elementos se moverán para llenar el vacío. Si hubiera más de una aparición de un valor, `remove()` sólo sacará la primera aparición.

Ordenar una lista

Las listas pueden ser ordenadas "in-place", es decir, sin usar nuevas variables.

```
s = [10, 1, 7, 3]
s.sort()                # [1, 3, 7, 10]

# Orden inverso
s = [10, 1, 7, 3]
s.sort(reverse=True)    # [10, 7, 3, 1]

# Funciona con cualquier tipo de datos que tengan orden
s = ['foo', 'bar', 'spam']
s.sort()                # ['bar', 'foo', 'spam']
```

Usá `sorted()` si querés generar una nueva lista ordenada en lugar de ordenar la misma:

```
t = sorted(s)           # s queda igual, t guarda los valores ordenado
```



Listas y matemática

Cuidado: Las listas no fueron diseñadas para realizar operaciones matemáticas.

```
>>> nums = [1, 2, 3, 4, 5]
>>> nums * 2
[1, 2, 3, 4, 5, 1, 2, 3, 4, 5]
>>> nums + [10, 11, 12, 13, 14]
[1, 2, 3, 4, 5, 10, 11, 12, 13, 14]
```

Específicamente, las listas no representan vectores ni matrices como en MATLAB, Octave, R, etc. Sin embargo, hay paquetes de Python que hacen muy bien ese trabajo (por ejemplo [numpy](#)).

Ejercicios

En este ejercicio, vamos a experimentar con el tipo de dato *lista* de Python. En la última sección, trabajaste con cadenas que hacían referencia a cajones de frutas.

```
>>> frutas = 'Frambuesa,Manzana,Naranja,Mandarina,Banana,Sandía,Pera'
```

Armá una lista con los nombres de frutas usando el comando `split()` :

```
>>> lista_frutas = frutas.split(',')
```

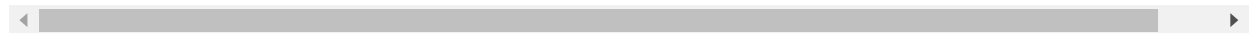
Ejercicio 1.22: Extracción y reasignación de elementos.

Probá un par de estos comandos para extraer un elemento:

```
>>> lista_frutas[0]
'Frambuesa'
>>> lista_frutas[1]
'Manzana'
>>> lista_frutas[-1]
'Pera'
>>> lista_frutas[-2]
'Sandía'
>>>
```

Intentá reasignar un valor:

```
>>> lista_frutas[2] = 'Granada'
>>> lista_frutas
['Frambuesa', 'Manzana', 'Granada', 'Mandarina', 'Banana', 'Sandía', 'Pera']
>>>
```



Hacé unas rebanadas (slices) de la lista:

```
>>> lista_frutas[0:3]
['Frambuesa', 'Manzana', 'Granada']
>>> lista_frutas[-2:]
['Sandía', 'Pera']
>>>
```

Creá una lista vacía y agregale un elemento.

```
>>> compra = []
>>> compra.append('Pera')
>>> compra
['Pera']
```

Podés incluso reasignar una lista a una porción de otra lista. Por ejemplo:

```
>>> lista_frutas[-2:] = compra
>>> lista_frutas
['Frambuesa', 'Manzana', 'Granada', 'Mandarina', 'Banana', 'Pera']
>>>
```

Cuando hacés esto, la lista del lado izquierdo (`lista_frutas`) va a cambiar su tamaño para que encaje la lista del lado derecho (`compra`). En el ejemplo de arriba los últimos dos elementos de la `lista_frutas` fueron reemplazados por un solo elemento en la lista `compra` .

Ejercicio 1.23: Ciclos sobre listas

El ciclo `for` funciona iterando sobre datos en una secuencia. Antes vimos que podíamos iterar sobre los caracteres de una cadena (las cadenas son secuencias). Ahora veremos que podemos iterar sobre listas también. Verificá esto tipeando lo que sigue y viendo qué pasa:

```
>>> for s in lista_frutas:
    print('s =', s)
```

Ejercicio 1.24: Test de pertenencia

Usá los operadores `in` o `not in` para verificar si `'Granada'` , `'Lima'` , y `'Limon'` pertenecen a la lista de frutas.

```
>>> # ¿Está 'Granada' IN `lista_frutas`?
True
>>> # ¿Está 'Lima' IN `lista_frutas`?
False
>>> # ¿Está 'Limon' NOT IN `lista_frutas`?
True
>>>
```

Ejercicio 1.25: Adjuntar, insertar y borrar elementos

Usá el método `append()` para agregar `'Mango'` al final de `lista_frutas` .

```
>>> # agregar 'Mango'
>>> lista_frutas
['Frambuesa', 'Manzana', 'Granada', 'Mandarina', 'Banana', 'Pera', 'Mango']
>>>
```

Usá el método `insert()` para agregar `'Lima'` como segundo elemento de la lista.

```
>>> # Insertar 'Lima' como segundo elemento
>>> lista_frutas
['Frambuesa', 'Lima', 'Manzana', 'Granada', 'Mandarina', 'Banana', 'Pera',
>>>
```

Usá el método `remove()` para borrar `'Mandarina'` de la lista.

```
>>> # Borrar 'Mandarina'
>>> lista_frutas
['Frambuesa', 'Lima', 'Manzana', 'Granada', 'Banana', 'Pera', 'Mango']
>>>
```

Agregá una segunda copia de `'Banana'` al final de la lista.

Observación: es perfectamente válido tener valores duplicados en una lista.

```
>>> # Agregar 'Banana'
>>> lista_frutas
['Frambuesa', 'Lima', 'Manzana', 'Granada', 'Banana', 'Pera', 'Mango', 'Ba
>>>
```



Usá el método `index()` para determinar la posición de la primera aparición de `'Banana'` en la lista.

```
>>> # Encontrar la primera aparición de 'Banana'
>>> lista_frutas
4
>>> lista_frutas[4]
'Banana'
>>>
```

Contá la cantidad de apariciones de `'Banana'` en la lista:

```
>>> lista_frutas.count('Banana')
2
>>>
```

Borrá la primera aparición de `'Banana'` .

```
>>> # Borrar la primer aparición de 'Banana'
>>> lista_frutas
['Frambuesa', 'Lima', 'Manzana', 'Granada', 'Pera', 'Mango', 'Banana']
>>>
```

Para que sepas, no hay un método que permita encontrar o borrar *todas* las apariciones de un elemento en una lista. Más adelante veremos una forma elegante de hacerlo.

Ejercicio 1.26: Sorting

¿Querés ordenar una lista? Usá el método `sort()`. Probalo:

```
>>> lista_frutas.sort()
>>> lista_frutas
['Banana', 'Frambuesa', 'Granada', 'Lima', 'Mango', 'Manzana', 'Pera']
>>>
```

¿Y si ordenamos al revés?

```
>>> lista_frutas.sort(reverse=True)
>>> lista_frutas
['Pera', 'Manzana', 'Mango', 'Lima', 'Granada', 'Frambuesa', 'Banana']
>>>
```

Observación: acordate de que el método `sort()` modifica el contenido de la misma lista *in-place*. Los elementos son reordenados moviéndolos de una posición a otra, pero no se crea una nueva lista.

Ejercicio 1.27: Juntar múltiples cadenas

Si querés juntar las cadenas en una lista, usá el método `join()` de los strings como sigue (ojo: parece un poco raro al principio).

```
>>> lista_frutas = ['Banana', 'Mango', 'Frambuesa', 'Pera', 'Granada', 'Ma
>>> a = ','.join(lista_frutas)
>>> a
'Banana,Mango,Frambuesa,Pera,Granada,Manzana,Lima'
>>> b = ':'.join(lista_frutas)
>>> b
'Banana:Mango:Frambuesa:Pera:Granada:Manzana:Lima'
>>> c = ''.join(lista_frutas)
>>> c
'BananaMangoFrambuesaPeraGranadaManzanaLima'
>>>
```



Ejercicio 1.28: Listas de cualquier cosa

Las listas pueden contener cualquier tipo de objeto, incluyendo otras listas (serían 'listas anidadas').

Probá esto:

```
>>> nums = [101, 102, 103]
>>> items = ['spam', lista_frutas, nums]
>>> items
['spam', ['Banana', 'Mango', 'Frambuesa', 'Pera', 'Granada', 'Manzana', 'L
```


Fijate bien el output. `items` es una lista con tres elementos. El primero es un string, pero los otros dos elementos son listas.

Podés acceder a los elementos de las listas anidadas usando múltiples operaciones de acceso por índice.

```
>>> items[0]
'spam'
>>> items[0][0]
's'
>>> items[1]
['Banana', 'Mango', 'Frambuesa', 'Pera', 'Granada', 'Manzana', 'Lima']
>>> items[1][1]
'Mango'
>>> items[1][1][2]
'n'
>>> items[2]
[101, 102, 103]
>>> items[2][1]
102
>>>
```

A pesar de que es técnicamente posible hacer una estructura de listas muy complicada, como regla general, es mejor mantener las cosas simples. Lo más usual es guardar en las listas muchos elementos del mismo tipo. Por ejemplo, una lista sólo de números o una lista de cadenas. Mezclar diferentes tipos de datos en una misma lista puede volverse conceptualmente difuso, así que mejor lo evitamos.

Ejercicio 1.29: Traductor (rústico) al lenguaje inclusivo

Queremos hacer un traductor que cambie las palabras masculinas de una frase por su versión neutra. Como primera aproximación, completá el siguiente código para reemplazar todas las letras 'o' que figuren en el último o anteúltimo carácter de cada palabra por una 'e'. Por ejemplo 'todos somos programadores' pasaría a ser 'todes somes programdores'. Guardá tu código en el archivo `inclusive.py`

```
>>> frase = 'todos somos programadores'
>>> palabras = frase.split()
>>> for palabra in palabras:
    if ?
    ...
    frase_t = ?
    print(frase_t)
'todes somes programdores'
>>>
```

Probá tu código con 'Los hermanos sean unidos porque ésta es la ley primera', '¿cómo transmitir a los otros el infinito Aleph?' y 'Todos, tu también'. ¿Qué falla en esta última? (¡no hace falta que lo resuelvas!)

[Contenidos](#) | [Anterior \(4 Cadenas\)](#) | [Próximo \(6 Cierre de la primer clase\)](#)