Listas

October 22, 2020

1 Listas

Las listas son secuencias mutables, que se utilizan para almacenar colecciones de items. Hereda de la clase iterable .

[5]:
```
# ejemplos
variable = [1, 2, 3, 4, "hola", [2,3,4.5], None]

for v in variable:
    print(v)

variable[4] = "chaauuu"
print(variable)
```

```
1
2
3
4
hola
[2, 3, 4.5]
None
[1, 2, 3, 4, 'chaauuu', [2, 3, 4.5], None]
```

1.1 Slices

[16]:
```
# ejemplos
for i in variable[2:10:2]:
    print(i)
```

```
3
chaauuu
None
```

1.2 del

elimina un elemento

[53]:
```
a = [1,23, 4,"hola", False]
del a[1]
```

[54]:
```
print(a)
```

```
[1, 4, 'hola', False]
```

1.3 Append

Agrega una item al final de la lista

[24]:
```
variable.append("emma")
variable.append([1,2,3])
```

[25]:
```
print(variable)
```

```
[1, 2, 3, 4, 'chaauuu', [2, 3, 4.5], None, 'emma', 'emma', 2, 'emma', 2.4,
'emma', False, 'emma', [1, 2, 3]]
```

1.4 extend

Extender una lista con otra secuencias

[30]:
```
variable.extend((2,1,3))
```

[31]:
```
print(variable)
```

```
[1, 2, 3, 4, 'chaauuu', [2, 3, 4.5], None, 'emma', 'emma', 2, 'emma', 2.4,
'emma', False, 'emma', [1, 2, 3], 2, 1, 3, 2, 1, 3]
```

1.5 insert

Insertar un elemento en una posición determinada.

[34]:
```
variable.insert(4, ["otro valor"])
```

[35]:
```
variable
```

[35]:
```
[1,
 2,
 3,
 4,
 ['otro valor'],
 'otro valor',
 'chaauuu',
 [2, 3, 4.5],
 None,
 'emma',
 'emma',
 2,
 'emma',
 2.4,
```

```
'emma',
False,
'emma',
[1, 2, 3],
2,
1,
3,
2,
1,
3]
```

1.6 remove

Elimina el primer elemento de la lista que coincida con el argumento enviado.

[38]:
```
try:
    variable.remove('algo que no esta')
except ValueError:
    print("no existe")
```

```
no existe
```

1.7 pop

Elimina un elemento de la lista en una posición determinada

[42]:
```
print(variable.pop(4))
print(variable)
```

```
otro valor
[1, 2, 3, 4, 'chaauuu', [2, 3, 4.5], None, 'emma', 2, 'emma', 2.4, 'emma',
False, 'emma', [1, 2, 3], 2, 1, 3, 2]
```

1.8 clear

Elimina todos los elementos de una lista

[45]:
```
variable.clear()
```

[46]:
```
variable
```

[46]:
```
[]
```

[68]:
```
variables = [1, 2, 3, 4, 'chaauuu', [2, 3, 4.5], None, 'emma', 2, 'emma', 2.4,␣
↪'emma', False, 'emma', [1, 2, 3], 2, 1, 3, 2]
id(variables)
```

[68]:
```
140356848518016
```

[ ]:

[66]:
```
variables = []
```

[67]:
```
id(variables)
```

[67]:
```
140356848640576
```

[ ]:

1.9 count

Devuelve el número de veces que aparece un elemento determinado en la lista

[75]:
```
variables.count([1, 2, 3])
```

[75]:
```
1
```

1.10 index

Retorna el indice de la primera ocurrencia de un elemento dado

[85]:
```
variables.index('emma')
variables.index('emma', 0, 5)
```

```
␣
↪---------------------------------------------------------------------------

ValueError                                Traceback (most recent call␣
↪last)

<ipython-input-85-5902a171dd0c> in <module>
  1 variables.index('emma')
----> 2 variables.index('emma', 0, 5)


ValueError: 'emma' is not in list
```

1.11 sort

Ordena la lista. Cuidado: Ordena la lista en sí, no genera copia.

[103]:
```
variables = [1, 2, 3, 4, 6, -52, -3, -2]
```

[101]:
```
# variables.sort(reverse=True)
```

[102]:
```
variables
```

[102]:
```
[6, 4, 3, 2, 1, -2, -3, -52]
```

[104]:
```
var2 = sorted(variables, reverse=True)
```

[105]:
```
variables
```

[105]:
```
[1, 2, 3, 4, 6, -52, -3, -2]
```

[106]:
```
var2
```

[106]:
```
[6, 4, 3, 2, 1, -2, -3, -52]
```

[111]:
```
def comp(x):
    return x['cant_h']
```

[112]:
```
foo = [[6, 10], [3, 100], [10, 600]]

sorted(foo, key=comp, reverse=True)
```

[112]:
```
[[10, 600], [3, 100], [6, 10]]
```

1.12 reverse

"Doy vuelta" una lista.

[123]:
```
variables.reverse()
```

[122]:
```
variables
```

[122]:
```
[-2, -3, -52, 6, 4, 3, 2, 1]
```

[ ]:
```

```

1.13 copy

Creo una copia

[125]:
```
a = variables
```

[126]:
```
variables
```

[126]:
```
[1, 2, 3, 4, 6, -52, -3, -2]
```

[127]:
```
a
```

[127]:
```
[1, 2, 3, 4, 6, -52, -3, -2]
```

[128]:
```
a[0] = "holaa"
```

[129]:
```
print(a)
```

```
['holaa', 2, 3, 4, 6, -52, -3, -2]
```

[130]:
```
variables
```

[130]:
```
['holaa', 2, 3, 4, 6, -52, -3, -2]
```

[131]:
```
id(a)
```

[131]:
```
140356848479168
```

[132]:
```
id(variables)
```

[132]:
```
140356848479168
```

[134]:
```
var = variables.copy()
```

[135]:
```
var
```

[135]:
```
['holaa', 2, 3, 4, 6, -52, -3, -2]
```

[137]:
```
var[0] = 'chau'
```

[138]:
```
print(var)
```

```
['chau', 2, 3, 4, 6, -52, -3, -2]
```

[139]:
```
print(variables)
```

```
['holaa', 2, 3, 4, 6, -52, -3, -2]
```

[140]:
```
id(variables)
```

[140]:
```
140356848479168
```

[141]:
```
id(var)
```

[141]:
```
140356849131136
```

1.14 zip

zippea listas

[143]:
```
paises = ['Arg', 'Bolivia', 'Peru']
capitales = ['BsAs', 'LP', 'LIMA']
```

[155]:
```
for p, c in zip(paises, capitales):
    print(f'{p}, {c}')
```

```
Arg, BsAs
Bolivia, LP
Peru, LIMA
```

1.15 Ejercicio

- Implementar rápidamente una cola (FIFO) y una pila (LIFO).

[47]:
```
# ejercicio resuelto
```

1.16 Lista de Compresión (List Comprehensions)

Proveen una forma concisa de crear listas. Se las suele utilizar cuando creamos una lista donde cada elemento es resultado de otra operación aplicado a elementos de otra secuencia o crear secuencias que que cada elemento cumpla con una condición.

[161]:
```
# una lista de cuadrados
resultado = []
bases = [2, 4, 6, 7, 8]

for b in bases:
    if b ** 2 > 30:
        resultado.append(b**2)
```

[162]:
```
print(resultado)
```

```
[36, 49, 64]
```

[163]:
```
# usando list comprehensions
resultado = [b ** 2 for b in bases if b**2 > 30]
```

[164]:
```
resultado
```

[164]:
```
[36, 49, 64]
```

[170]:
```
# Convertir lista de lista en una sola lista
vec = [[1,2,3], [4,5,6], [7,8,9]]
# [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```python
result = []

for n in vec:
    for x in n:
        result.append(x)

result2 = [x for n in vec for x in n]

# Mismo, pero que sean todos pares?

result2 = [x for n in vec for x in n if x % 2 == 0]
```

[171]:
```python
print(result2)
```

```
[2, 4, 6, 8]
```

1.17 Ejercicios

- Combinar dos listas de a pares sin que los elementos se repitan (List comprehensions)

input: `a = [1, 2, 3]; b = [3, 4, 5]`

output = `[(1, 3), (1, 4), (1, 5), (2, 3), (2, 4), ...]`

- Calcular la traspuesta de:

```
matrix = [
    [1, 2, 3, 4],
    [5, 6, 7, 8],
    [9, 10, 11, 12],
]

output = [
    [1, 5, 9],
    [2, 6, 10],
    [3, 7, 11],
    [4, 8, 12],
]
```

- Desarrollar un un script que me liste el contenido de archivos de una carpeta con sus pesos (bytes, kb , mb, etc) y copiar los archivos a otra capeta cambiandole nombre y extensión y el nombre a: `<nombre_del_achivo_sin_extension>_ejercicio_python.txt`

```
test/ * hola.txt
      * chau.py

hola.txt 0kb
chau.py 10kb


test2/ hola.txt_ejercicio_python.txt
```

chau.py_ejercicio_python.txt