Funciones y métodos de listas

Las listas en Python vienen con una colección de **operaciones** muy completa. Veamos algunas de ellas:

Rangos o secciones o slices

En Python hay distintos modos de extraer fragmentos de una lista. Imposible ecxplicarlo mejor que con un ejemplo.

In [1]:

```
lista = list(range(10, 101, 10))
print("a) ", lista)
print("b) ", lista[3:6])
print("c) ", lista[:6])
print("d) ", lista[6:])
print("e) ", lista[-1:0:-1])
print("f) ", lista[::-1])
print("g) ", lista[0:10:1])
print("h) ", lista[ :  :1])
print("i) ", lista[0:10:2])
print("j) ", lista[::-2])
```

```
a)  [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
b)  [40, 50, 60]
c)  [10, 20, 30, 40, 50, 60]
d)  [70, 80, 90, 100]
e)  [100, 90, 80, 70, 60, 50, 40, 30, 20]
f)  [100, 90, 80, 70, 60, 50, 40, 30, 20, 10]
g)  [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
h)  [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
i)  [10, 30, 50, 70, 90]
j)  [100, 80, 60, 40, 20]
```

Funciones de uso frecuente

Repasamos seguidamente algunas de las operaciones más frecuentes con listas.

```
lista.append(objeto)
    Añade un objeto al final de una lista

lista.count(valor) -> integer
    Da el número de apariciones de un valor en una lista


lista.extend(iterable)
    Extiende una lista, al final, añadiendo una colección de elementos (iterable)

lista.index(valor, [start, [stop]]) -> integer
    Da el primer índice en que se encuentra el valor
```

```
        Dispara un ValueError si el valor no está presente

    lista.insert(index, objeto)
        Inserta un objeto en la posición indicada

    lista.pop([index]) -> item
        Elimina (y devuelve) el elemento en la posición dada por index
        (Por defecto, el elemento será el último)
        Dispara un IndexError si la lista está vacía o el índice está fuera de rango

    lista.remove(valor)
        Elimina el primer elemento igual al valor dado
        Dispara un ValueError si el valor no está presente

    lista.reverse()
        invierte una lista *in place*

    lista.sort(cmp=None, key=None, reverse=False)
        sort *in place*
        cmp(x, y) -> -1, 0, 1
```

In [2]:

```
l = range(5)
l
```

Out[2]:

```
range(0, 5)
```

In [3]:

```
l.append(7) # deseamos añadir el elemento 7 al final de la lista... pero se produce un erro
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-3-afa9ad1b6fd8> in <module>
----> 1 l.append(7) # deseamos añadir el elemento 7 al final de la lista...
 pero se produce un error:

AttributeError: 'range' object has no attribute 'append'
```

In [4]:

```
l = list(range(5)) # Como range() no genera una lista, hay que convertirlo en una lista
l.append(7)        # Y ya funciona append, añadiendo el elemento 7 a la lista, *in place*
l
```

Out[4]:

```
[0, 1, 2, 3, 4, 7]
```

In [5]:

```
l.extend([45, 56, 3, 2])
# Este método extiende la lista, añadiendo al final
# todos los elementos dados en la lista parámetro
l
```

Out[5]:

```
[0, 1, 2, 3, 4, 7, 45, 56, 3, 2]
```

In [6]:

```
l = list(range(4))
l = l + [45, 56, 3, 2] # La concatenación crea una lista nueva
print(l)
l[4] = 200
print(l)
```

```
[0, 1, 2, 3, 45, 56, 3, 2]
[0, 1, 2, 3, 200, 56, 3, 2]
```

In [7]:

```
l = list(range(4))
print(l)
l.insert(0, 100)    # En cambio, insert opera *in place*
print(l)
```

```
[0, 1, 2, 3]
[100, 0, 1, 2, 3]
```

In [8]:

```
l = list(range(4))
l.insert(-1,100)
print(l)
l.insert(-1,200)
print(l)
```

```
[0, 1, 2, 100, 3]
[0, 1, 2, 100, 200, 3]
```

In [9]:

```
def posic(lst, elem):
    """
    Da la primera posición de un elemento en una lista
    Si el elemento no está en la lista, el resultado es -1

    Parameters
    ------------------------
    elem: Alpha, un tipo cualquiera
    lst: lista de Alpha


    Returns
    ---------------------------
    pos si, si elem está presente en lst[pos], y no antes
    -1, en caso contrario

    Example
    ---------------------------------------
    >>> posic([1, 2, 5, 45], 5)
    2
    """
    i = 0
    while i<len(lst) and lst[i]!=elem:
        i += 1
    if i==len(lst):
        return -1
    else:
        return i
```

In [10]:

```
l = list(range(5,15))
print(l)
print(posic(l, 7), posic(l, 20))
```

```
[5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
2 -1
```

In [11]:

```
print(l)
print(l.index(8)) # En realidad, existe un método index predefinido :-)
```

```
[5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
3
```

In [12]:

```
l
l.index(15) # pero si el elemento no está, el método index da un error
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-12-c05323e57243> in <module>
      1 l
----> 2 l.index(15) # pero si el elemento no está, el método index da un err
or

ValueError: 15 is not in list
```

Eliminación de elementos en una lista

In [13]:

```
l = [5, 6, 7, 8, 9, 10, 11, 12, 13]
print(l)
l.pop()
print(l)
```

```
[5, 6, 7, 8, 9, 10, 11, 12, 13]
[5, 6, 7, 8, 9, 10, 11, 12]
```

Podemos desear también eliminar todos los elementos entre dos posiciones dadas:

In [14]:

```
def eliminar(lst, ini, end):
    """
    Eliminación *in place* de todos los elementos lst[j], para ini <= j < end

    Parameters
    ----------
    lst: [x]
    ini: int
    end: int


    0 <= ini < end <= len(lst)

    Returns
    -------
    None

    Action
    ------
    modifica lst, eliminando todos los elementos entre ini y end

    Example
    -------
    >>> l = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    >>> eliminar(l, 3, 5)
    >>> print(l)
    [1, 2, 3, 6, 7, 8, 9, 10]
    """

    # Primero adelantamos todos los elementos tras la franja que se va a eliminar,
    # esto es, movemos los elementos lst[i], para end <= i < len(lst),
    # a las posiciones que empiezan en ini:

    pos_end = end
    pos_ini = ini
    while pos_end<len(lst):
        lst[pos_ini] = lst[pos_end]
        pos_ini += 1
        pos_end += 1

    # Ahora, eliminamos los end-ini elementos desde el final de la lista:
    for i in range(end-ini):
        lst.pop()
```

In [15]:

```
l = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
eliminar(l, 3, 5)
print(l)
```

```
[1, 2, 3, 6, 7, 8, 9, 10]
```

Como ya podíamos imaginar, una función tan útil como ésta ya está predefinida:

In [16]:

```
l = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(l)
del l[2]
print(l)
del l[2:7] # Borra los elementos entre la posición 3 y 5 (sin incluir), lo mismo que delete
print(l)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
[1, 2, 4, 5, 6, 7, 8, 9, 10]
[1, 2, 9, 10]
```

In [17]:

```
del l[2:]
l
```

Out[17]:

```
[1, 2]
```

In [18]:

```
del l[5:2]  # queda sin efecto, al tratarse de un rango vacío
l
```

Out[18]:

```
[1, 2]
```

In [19]:

```
def invierte(lst):
    """
    Permuta *in place* los elementos de la lista dada,
    dejándola en orden inverso al proporcionado

    Parameters
    ----------
    lst: [x]

    Returns
    -------
    None

    Example
    -------
    >>> lista = [1, 2, 5, 4]
    >>> invierte(lista)
    >>> lista

    """
    # Permutamos la primera mitad de los elementos de la lista
    # con los de la otra mitad
    i = 0
    for i in range(0, len(lst)//2):
        j = len(lst) - 1 - i
        lst[i], lst[j] = lst[j], lst[i]
     # Cuidado: esta lista NO devuelve valor alguno: permuta los elementos *in place*
```

In [20]:

```
l = list(range(10))
print(l)
invierte(l)
print(l)
print(invierte(l))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
None
```

Por supuesto, esta función también está definida:

In [21]:

```
l = list(range(10))
print(l)
l.reverse()
print(l)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

Función enumerate

Utilísima, como se irá viendo, es la siguiente función:

In [22]:

```
list(enumerate("aeiou"))
```

Out[22]:

```
[(0, 'a'), (1, 'e'), (2, 'i'), (3, 'o'), (4, 'u')]
```

Catálogo de funciones predefinidas sobre listas

In [23]:

```
help(list)
```

```
|      Return self!=value.
|
|  __ge__(self, value, /)
|      Return self>=value.
|
|  __getattribute__(self, name, /)
|      Return getattr(self, name).
|
|  __getitem__(...)
|      x.__getitem__(y) <==> x[y]
|
|  __gt__(self, value, /)
|      Return self>value.
|
|  __iadd__(self, value, /)
|      Implement self+=value.
|
|  __imul__(self, value, /)
|      Implement self*=value.
|
```